

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

MICROCOMPUTER BASED
INTERACTIVE DISPLAY SYSTEM

by

Francisco J. Mariategui C.

and

Ivan Nelson Hall Jr.

June 1979

Thesis Advisor:

G. Rahe

Approved for public release; distribution unlimited.

T189137

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Microcomputer Based Interactive Display System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1979
7. AUTHOR(s) Francisco J. Mariategui C. Ivan Nelson Hall Jr.		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE June 1979
		13. NUMBER OF PAGES 565
		15. SECURITY CLASS. (of this report) Unclassified
		16a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) microcomputer plasma display touch panel interactive display		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This study was undertaken to design and implement a microcomputer based interactive display system suitable for use as a shipboard tactical-situation display. The stand-alone system included two plasma display scopes, one microcomputer, one CRT and one line printer. The scope of the effort included the interface of the display system via a RS-232 data/link to a PDP-11/50 minicomputer in order to emulate the		

(20. ABSTRACT Continued)

shipboard tactical environment. Of major interest was the integration of the hardware components and the software developed in this study into a coherent alphanumeric and graphical display system.

Microcomputer Based
Interactive Display System

by

Francisco J. Mariategui C.
Lieutenant (jg), Peruvian Navy
B.S., Peruvian Naval Academy, 1974

and

Ivan Nelson Hall Jr.

B.S., Virginia Polytechnical Institute and State University, 1971

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1979

ABSTRACT

This study was undertaken to design and implement a microcomputer based interactive display system suitable for use as a shipboard tactical-situation display. The stand-alone system included two plasma display scopes, one microcomputer, one CRT and one line printer. The scope of the effort included the interface of the display system via a RS-232 data/link to a PDP-11/50 minicomputer in order to emulate the shipboard tactical environment. Of major interest was the integration of the hardware components and the software developed in this study into a coherent alphanumeric and graphical display system

TABLE OF CONTENTS

I.	INTRODUCTION -----	11
II.	INTRODUCTION TO THE PROBLEM -----	15
	A. PREFACE -----	15
	B. SCENARIO -----	15
	C. DESIGN OBJECTIVES -----	29
	D. TECHNOLOGICAL BASE -----	33
	E. DEVELOPMENT PLAN -----	35
III.	DISPLAY MODES -----	38
	A. GENERAL INFORMATION -----	38
	B. COMMAND OPTIONS: CORRECTNESS -----	38
	C. GRAPHICAL DISPLAY MODE -----	40
	D. ALPHANUMERIC DISPLAY MODE -----	42
	E. DATA ELEMENTS -----	42
IV.	HARDWARE COMPONENTS -----	46
	A. MICROCOMPUTER DEVELOPMENT SYSTEM -----	46
	B. PLASMA PANEL DESCRIPTION -----	47
	C. PLASMA SCOPE TOUCH-PANEL DESCRIPTION ---	51
	D. DATAMEDIA ELITE 2500 TERMINAL DESCRIPTION -----	52
	E. PDP-11/50 MINICOMPUTER DESCRIPTION -----	54
V.	MDS-PDP INTERFACE -----	57
	A. GENERAL INFORMATION -----	57
	B. SOFTWARE COMPONENTS: FUNCTIONAL DESCRIPTION -----	58
	1. MDSPDP PL/M-80 Program -----	58

	2. PDPRECEIVE C Program -----	59
	3. PDPSEND C Program -----	60
C.	INTERFACE DEFINITION -----	60
	1. PDP-11/50 -----	60
	2. Data Line -----	60
	3. MDS -----	60
D.	PDP-11/50 -- MDS INTERRUPT DRIVEN DATA TRANSMISSION -----	63
VI.	SOFTWARE OVERVIEW -----	65
	A. GENERAL INFORMATION -----	65
	B. DATA STRUCTURES -----	65
	C. MDS DISPLAY SYSTEM SOFTWARE -----	66
	D. PDP-11/50 SOFTWARE -----	66
VII.	SYSTEM DEVELOPMENT SUMMARY -----	68
	A. COMPUTER TO COMPUTER INTERFACE -----	68
	B. HARDWARE AND SOFTWARE COMPONENTS -----	69
	1. MDS Microcomputer -----	69
	2. PDP-11/50 Minicomputer -----	73
VIII.	CONCLUSIONS AND RECOMMENDATIONS -----	75
APPENDIX A.	OPERATOR'S MANUAL FOR THE DISPLAY SYSTEM -----	78
APPENDIX B.	DATA ELEMENTS DESCRIPTION -----	96
APPENDIX C.	MDS-PDP INTERFACE DETAILED DESCRIPTION -----	112
	A. DATA/LINK CABLE-CONNECTOR DESCRIPTION-	112
	B. MDS HARDWARE MODIFICATIONS -----	112
	C. PDP SOFTWARE MODIFICATIONS -----	114
	D. MDS INTERRUPT SYSTEM -----	116

E.	PDP-11/50 INTERRUPT SYSTEM -----	117
APPENDIX D.	SOFTWARE FUNCTIONAL DESCRIPTION -----	118
A.	MDS PL/M-80 MODULES -----	118
B.	PDP C PROCEDURES-----	143
APPENDIX E.	DESCRIPTION OF GRAPHIC DISPLAY TECHNOLOGY -----	151
A.	GENERAL INFORMATION -----	151
B.	PLASMA PANEL PHYSICAL DESCRIPTION ----	152
APPENDIX F.	INTERRUPT STRUCTURES -----	155
A.	BACKGROUND INFORMATION -----	155
B.	PRIORITY INTERRUPTS -----	160
	1. Polled Priority Interrupts -----	160
	2. Vectored Priority Interrupts -----	160
C.	8080 MICROPROCESSOR INTERRUPT METHOD -	162
APPENDIX G.	PLASMA SCOPE TOUCH-PANEL DESCRIPTION -	170
APPENDIX H.	DATAMEDIA ELITE 2500 TERMINAL DESCRIPTION -----	174
A.	ELITE 2500 SPECIFICATIONS -----	174
B.	DESCRIPTION OF CONTROL FUNCTIONS -----	175
C.	STANDARD FEATURES -----	179
D.	OPTIONAL FEATURES -----	180
APPENDIX I.	OPERATION OF THE MDSPDP PROGRAM -----	181
APPENDIX J.	PROGRAM LISTINGS -----	187
	LIST OF REFERENCES -----	562
	INITIAL DISTRIBUTION LIST -----	565

LIST OF TABLES

1. Command Options -----	39
--------------------------	----

LIST OF FIGURES

1.	Display System Block Diagram -----	13
2.	System Block Diagram -----	14
3.	Initialization of Date and Time -----	18
4.	General Contact Characteristics -----	19
5.	Graphical Presentation of Plasma Scopes -----	20
6.	Command Options at CRT Screen -----	21
7.	Primary Plasma Scope -----	23
8.	Primary Plasma Display -----	24
9.	Secondary Plasma Scope -----	25
10.	Secondary Plasma Display -----	26
11.	Touch-Panel Output at Secondary Plasma Scope ---	27
12.	Display of Threats Relative to Own Ship -----	28
13.	Specific Contact Query and Result -----	30
14.	Hardware Reassignment -----	31
15.	Illustration of General Contact Statistics -----	44
16.	Illustration of Specific Contact Statistics -----	45
17.	MDS Microcomputer Configuration -----	48
18.	Plasma Scope Panel Construction -----	50
19.	Datamedia Elite 2500 Video Terminal -----	53
20.	PDP-11/50 Minicomputer Configuration -----	56
21.	MDS Data Link I/O Configuration -----	61
22.	Data Structures -----	67
23.	Data Link Pin Connections -----	113
24.	Interrupt Process Scheme -----	161

25.	8080 Example Interrupt Flow Diagram -----	166
26.	Touch-Panel Block Diagram -----	173

I. INTRODUCTION

The purpose of this study is to design and implement a microcomputer based interactive display system suitable for use as a shipboard tactical display. The basic display system consists of a microcomputer, 2 plasma display scopes, 1 plasma touch panel, 1 CRT and 1 line printer (Fig. 1). The shipboard tactical environment is simulated by interfacing the display system to a PDP-11/50 minicomputer which emulates an information source (Fig. 2).

The display system is designed to support data reception, remote processing (manipulation and handling of data to build the local data bases), information display of graphical and alphanumeric data and a Man-Machine Interface. The environment emulated at the PDP Minicomputer provides a data subset of the Naval Tactical Data System (NTDS), [Ref. 19]. The subset of information is provided via the PDP interface to the display system. The information consists of the Surface contact profile subset of the NTDS.

Chapter II contains the introduction to the problem and the development strategy. Chapter III contains a discussion of the different display modes implemented in the system. The hardware components are discussed in Chapter IV. A detailed description of the interface between both computers is presented in Chapter V. An overview of all the software developed is given in Chapter VI and in Chapter VII the

development process of this study is summarized. The final conclusions and recommendations of this study are presented in Chapter VIII.

DISPLAY SYSTEM BLOCK DIAGRAM .

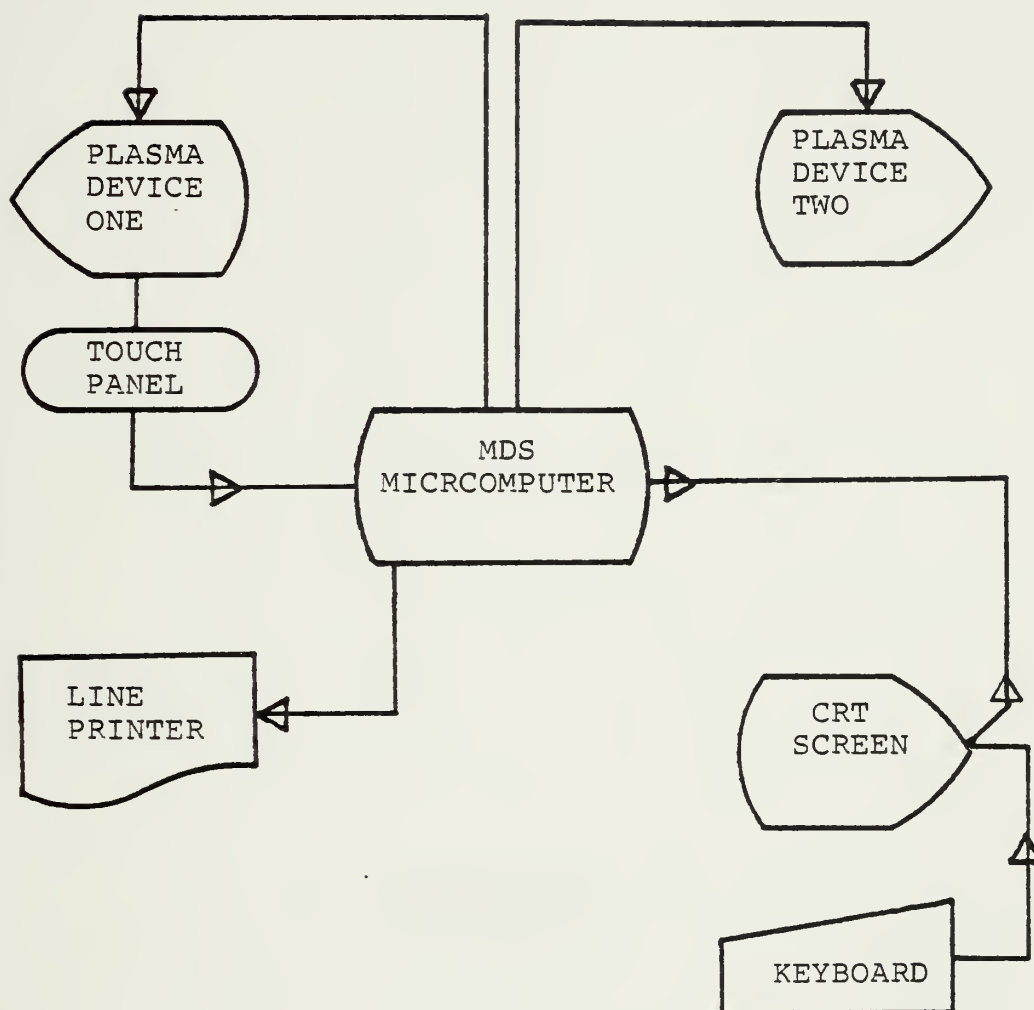


FIGURE 1

SYSTEM BLOCK DIAGRAM

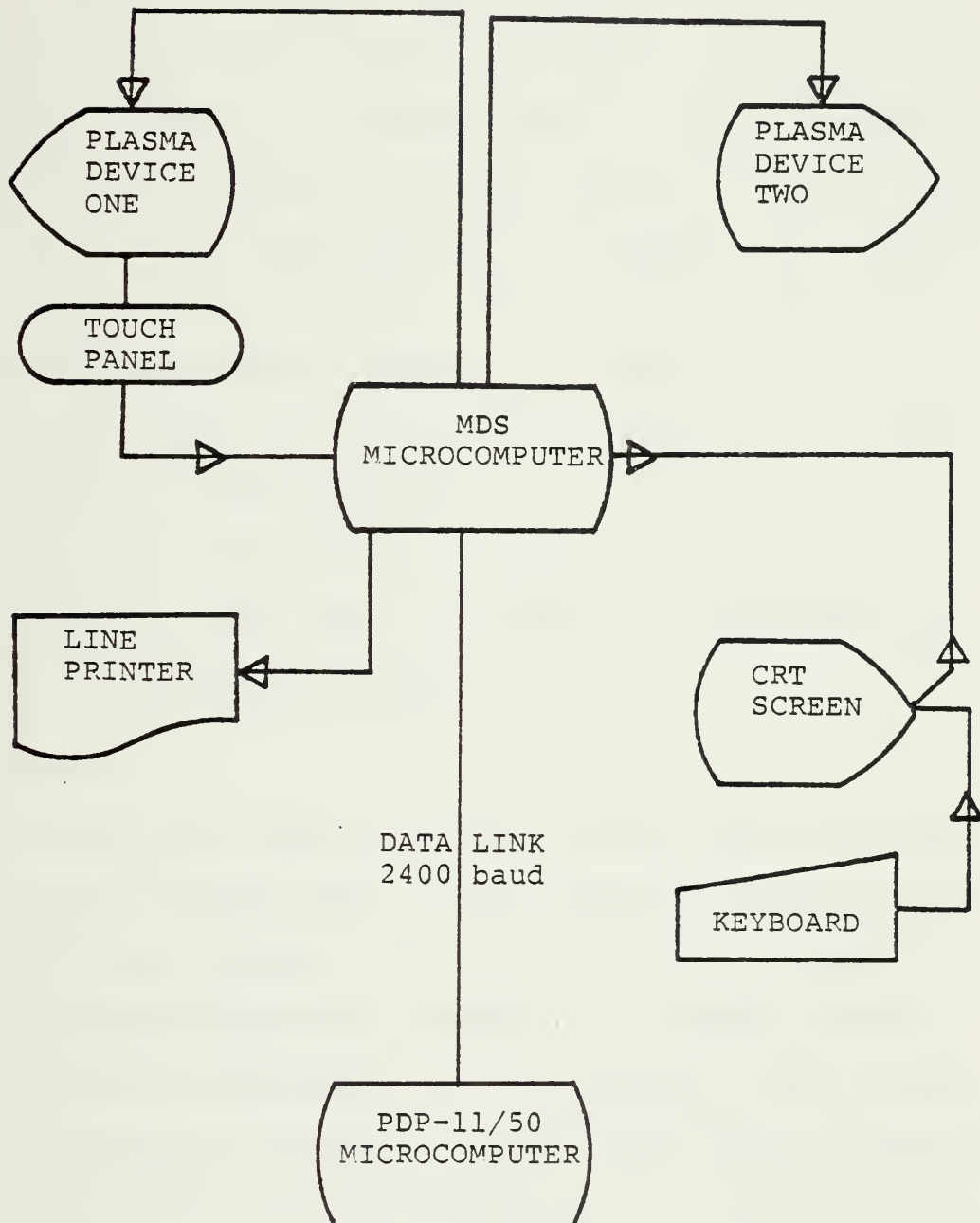


FIGURE 2

II. INTRODUCTION TO THE PROBLEM

A. PREFACE

In order to accomplish the Display of a shipboard tactical situation, microcomputer and plasma-display technologies were chosen as a technical base to design a "microcomputer based interactive display system." The shipboard tactical situation was provided by a hardware and software interface between the display system microcomputer and a PDP-11/50 (Minicomputer technology). The PDP was responsible for emulating the appropriate environment for the display system via the interface.

This study deals with the design and implementation of the "display system," the "interface," and the establishment of a Tactical environment.

B. SCENARIO

In order for a ship to maintain operational readiness as a unit of a task force, it must be aware of the current operational environment. The operational environment is defined as the subsurface, surface and airborne contact profiles in the geographic area of interest. The contact profile consists of friendly, hostile, and unknown contacts with associative contact characteristics. Contact characteristics are such data as latitude, longitude, course, speed, range, bearing, etc.

In typical task force situations, the command unit has the responsibility for the collection, analysis and dissemination of all information defining the operational environment. If a unit has NTDS capability, the information describing the operational environment is provided by the command unit via a radio link. The unit relies on the mainframe computer of the command unit for all information, analysis and two way communications. If a unit is a non-NTDS ship, the unit still relies on the mainframe computer of the command unit for information, but only through a one way teletype link. No analysis capability is provided. Once the non-NTDS unit receives the information, the data is manually plotted and analyzed. The procedure is not only time consuming but also requires from one to four watch personnel depending on the unit's condition of readiness.

This study demonstrates the feasibility of implementing an interactive display system using microcomputer and plasma display technologies for non-NTDS type ships. The display system implemented provides the capability to present a surface contact profile which constitutes the operational scenario on which the display system is based. For the purpose of this study, a PDP-11/50 minicomputer was utilized to emulate the information source or main frame computer of a command ship in a task force.

The operational scenario established for the study consisted of the PDP-11/50 computer generating a surface contact profile data base for a geographic region. The data

base is updated and transmitted to the display system in two minute intervals. Upon the reception of the data, the display system has the responsibility for building the local data base.

The data base constitutes the information that is presented at the display system in alphanumeric and graphical modes. The display system has the capability to allow operator interaction to query the system for presentation of specific contact alphanumeric or graphical characteristics relative to "ownship." Once the operator obtains the characteristics of a contact of interest, he can obtain a hard-copy via the display system teletype for dissemination.

The initialization of the display system begins with the setting of the system real-time clock and date. The operator enters: year, month, day, hours, minutes and seconds. The CRT screen presents the operator input as shown in figure 3.

Upon completion of the entry of the time and date of a tactical situation, the display system enters a ready state for the reception of data or operator interaction.

After the reception of a data set, the display system presents at the CRT screen the general contact characteristics as shown in figure 4. The general contact characteristics are also presented in a graphical format at the plasma scopes as shown in figure 5.

At any time the operator can request a list of command options to be presented at the CRT screen as shown in figure 6. The command options are the medium the operator

TIME: 00:28:41 DATE: 00/00/00 CONTACT # U1 STATISTICS		ONSHIP STATISTICS	
COURSE	: 334 DGS	COURSE	: 315 DGS
SPEED	: 20 KTS	SPEED	: 40 KTS
BEARING	: 073S DGS	LATITUDE	: 55.9N DGS
RANGE	: 051050 YDS	LONGITUDE	: 058.7W DGS
CPA TIME	: 00.00 H.M	QUADRANT NUMBER:	11
CPA DISTANCE	: 000000 YDS		
LATITUDE	: 48.8N DGS		
LONGITUDE	: 062.7W DGS		
COLLISION STATUS:			
QUADRANT NUMBER:	08		

INPUT DATE AND TIME.
YEAR: 79 MONTH: 06 DAY: 04 HOURS: 21 MINUTES: 13 SECONDS: 00
IS INPUT DATA CORRECT? (Y/N) _

INITIALIZATION OF DATE AND TIME

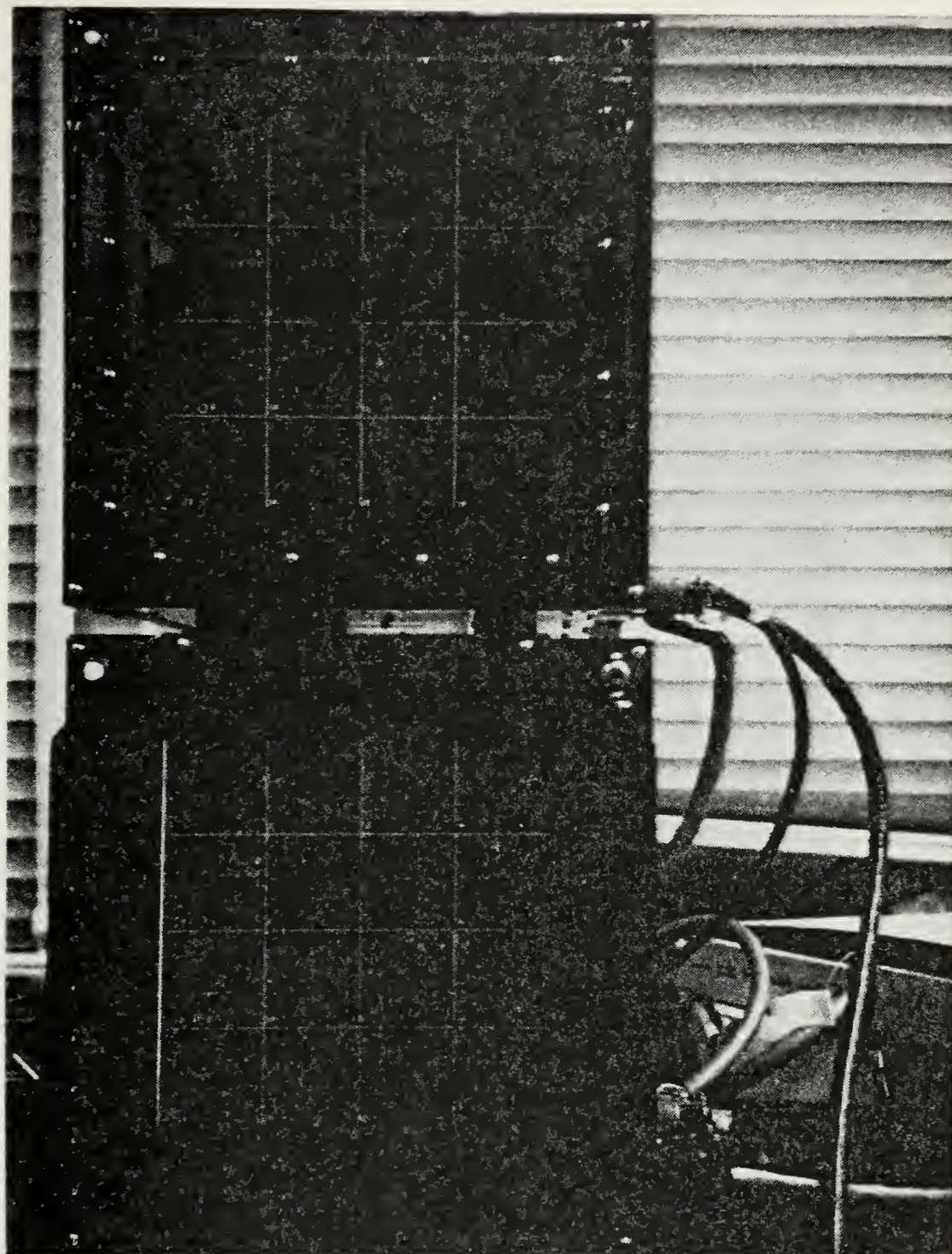
FIGURE 3

TIME: 00:21:15		DATE: 00/00/00					
CONTACT	QUADRANT	STATUS	TYPE	COURSE	SPEED	BEARING	RANGE
F0	12	FR		315 DG	40 KTS	135S DG	009370YD
F1	12	FR		315 DG	40 KTS	000 DG	000000YD
F2	12	FR		315 DG	40 KTS	135S DG	018750YD
H0	06	HD		315 DG	30 KTS	018P DG	075880YD
H1	06	HD		315 DG	30 KTS	020P DG	082650YD
H2	10	HD		315 DG	30 KTS	024P DG	080280YD
H3	10	HD		315 DG	30 KTS	022P DG	073290YD
U0	09	UN		180 DG	20 KTS	039P DG	114500YD
U1	08	UN		334 DG	20 KTS	066S DG	051240YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD

DATA UPDATED : READY .

GENERAL CONTACT CHARACTERISTICS

FIGURE 4



GRAPHICAL PRESENTATION AT PLASMA SCOPES

FIGURE 5

COMMAND OPTIONS

B.....: DISPLAY GENERAL DATA STATISTICS TO CRT SCREEN.
G.....: DISPLAY GENERAL DATA STATISTICS TO LINE PRINTER.
S.....: DISPLAY GENERAL DATA STATISTICS PER DATA RECEPTION.
L-F, L-H, L-U...: DISPLAY CONTACT STATISTICS TO SCREEN.
P-F, P-H, P-U...: DISPLAY CONTACT STATISTICS TO LINE PRINTER.
M.....: DISPLAY CURRENT TIME AND DATE TO SCREEN.
T.....: RESET DATE AND TIME.
R.....: DATA RECEPTION VERIFICATION TO SCREEN.
Z.....: SET STATUS OF PLASMA (1 AND 2) AND LINE PRINTER.
M.....: DISPLAY COMMAND OPTIONS TO SCREEN.
M.....: DISPLAY COMMAND OPTIONS TO LINE PRINTER.

COMMAND OPTIONS AT CRT SCREEN

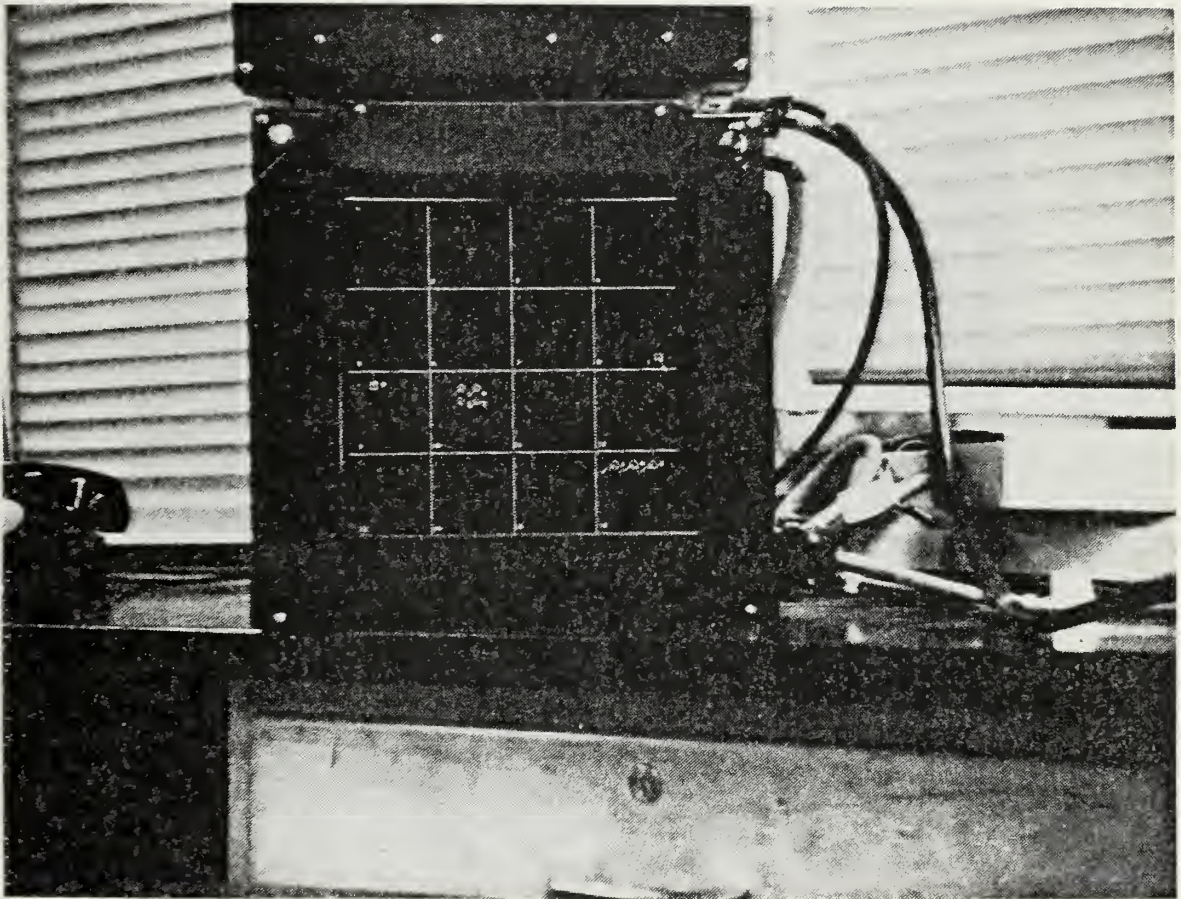
FIGURE 6

uses to interact with the display system. The operator may request general or specific contact data, set display modes, and initialize or shutdown display system hardware components.

The graphical data display at the plasma scopes is presented in a primary and secondary type of display. The primary plasma scope presents the surface contact profile of interest showing the symbolic representation of contacts with vector tails representing contact speed and course. The primary plasma scope is shown in figure 7 and the primary plasma display is shown in figure 8. The secondary plasma scope presents the surface contact profile of interest showing just the symbolic representation of the contacts. The secondary plasma scope is shown in figure 9 and the secondary plasma display is shown in figure 10.

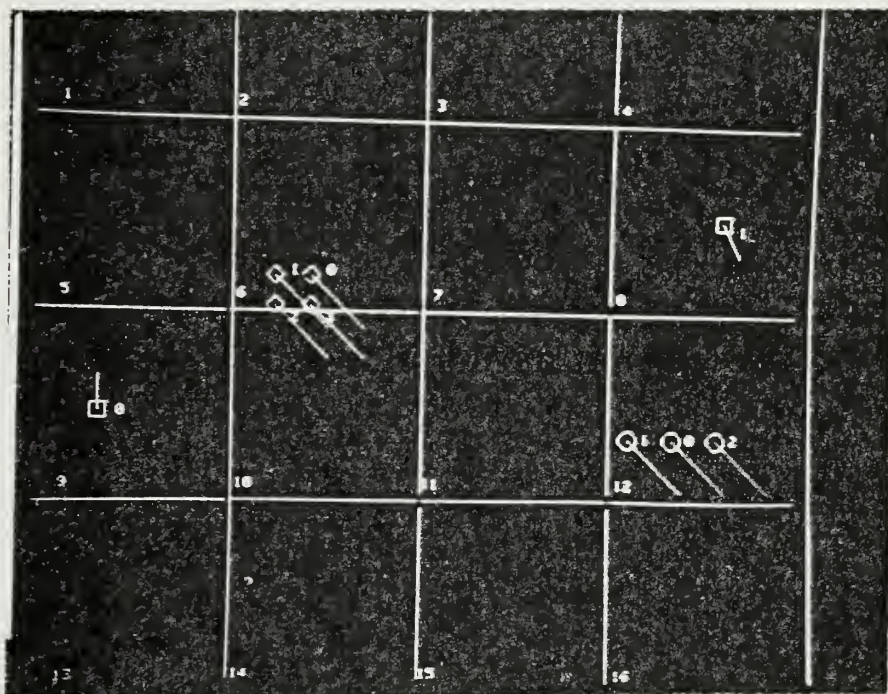
The operator has the responsibility to analyze the data presented, interact with the display system as necessary, to determine both the overall surface contact profile and any possible threat to "ownship." The analysis is performed by utilizing the primary plasma display as the main source of information. The visual information presented provides the operator with the necessary information to take action.

The action taken, if any, develops in the following logical sequence. The operator identifies a potential threat at the primary plasma display. The operator then uses the touch-panel feature to plot the threat relative to "ownship" at the secondary plasma scope (see figure 11). Figure 12 illustrates the secondary plasma display of the plot showing the threat's position relative to "ownship."



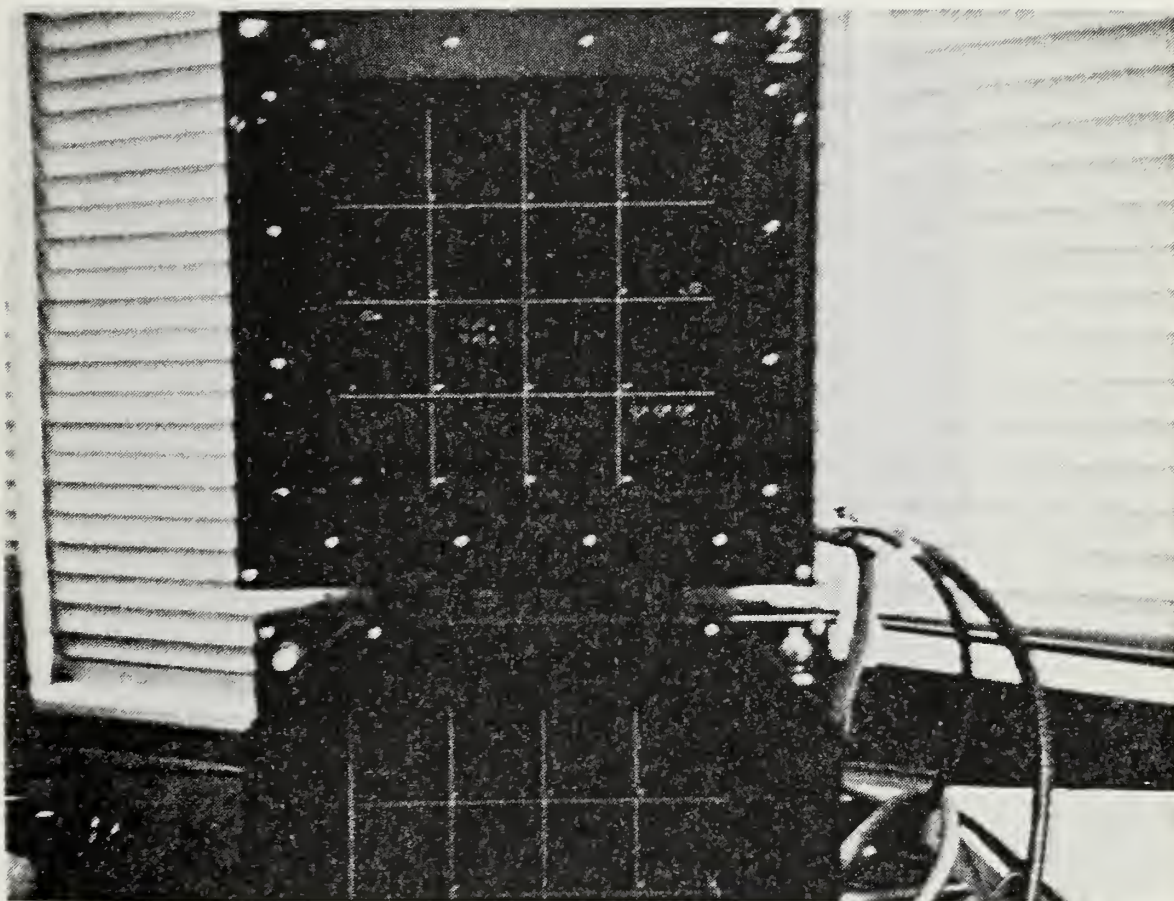
PRIMARY PLASMA SCOPE

FIGURE 7



PRIMARY PLASMA DISPLAY

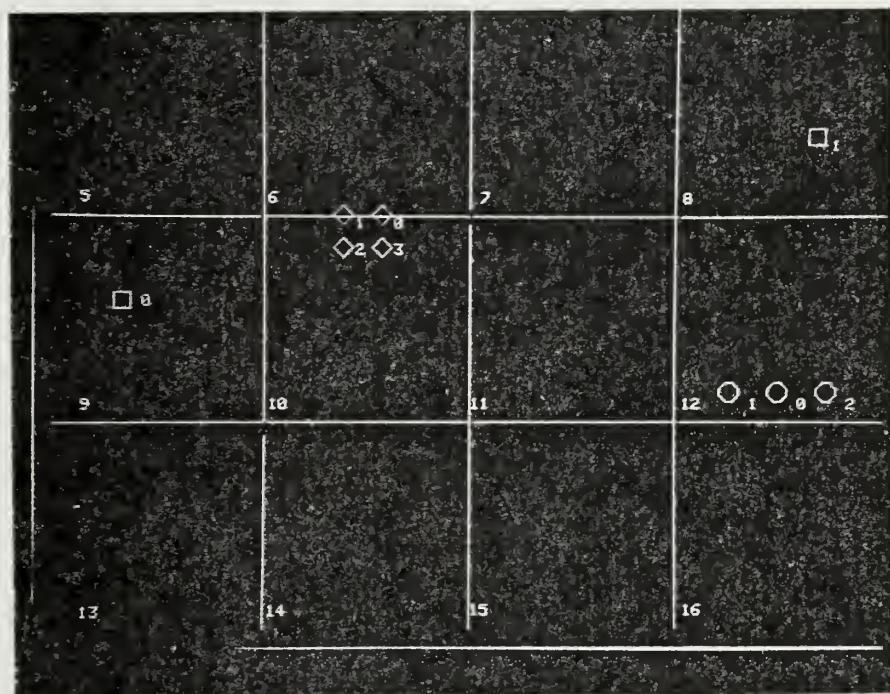
FIGURE 8



SECONDARY PLASMA SCOPE

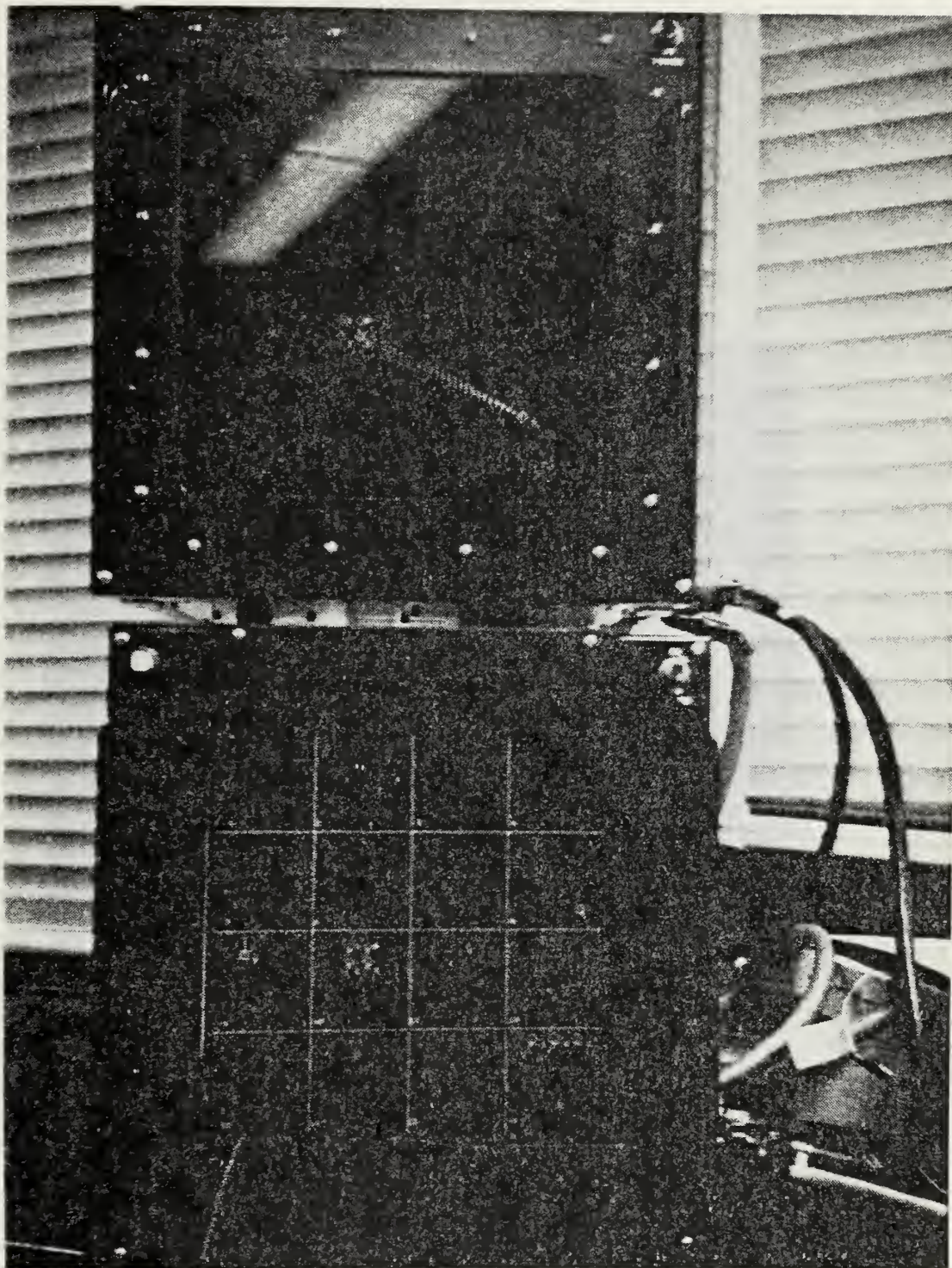
FIGURE 9

P.3



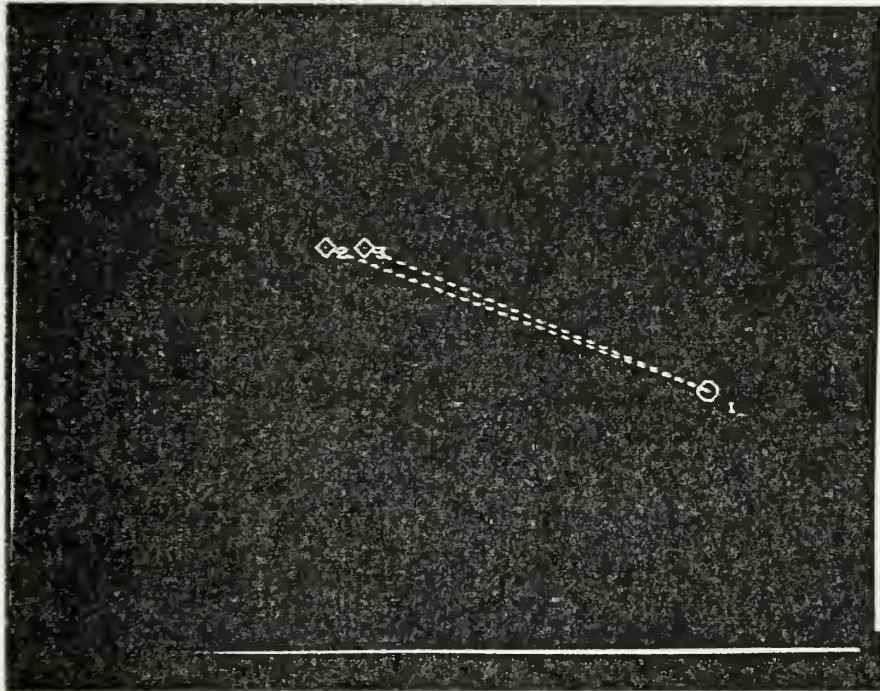
SECONDARY PLASMA DISPLAY

FIGURE 10



TOUCH-PANEL OUTPUT AT SECONDARY PLASMA SCOPE

FIGURE 11



DISPLAY OF THREATS RELATIVE TO 'OWNSHIP'

FIGURE 12

At this point the operator is able to zoom-in on the specific threat contact's characteristics by executing a command option. Figure 13 illustrates the execution of the command option and a possible result at the CRT screen.

At any time the operator can obtain or request a hard-copy of the most recent general or specific contact characteristics for dissemination. The format of the hardcopies is exactly the same as the ones presented at the CRT screen. The operator may also at any time reset time and date, and select the display mode and hardware device. The execution of the command to reset hardware devices is shown in figure 14.

C. DESIGN OBJECTIVES

The graphical and alphanumeric type of information to be displayed at the display system will be a mix of inter-related graphical and alphanumeric contact characteristics. This information reflects the Surface Contact Profile of any 80 by 80 square mile area, with the maximum capability of displaying information of 10 Friendly contacts, 10 Hostile contacts, and 10 Unknown contacts simultaneously. The 80 by 80 square mile area will be divided into 20 by 20 square mile quadrants to provide for visualization and a basis for operation interaction.

The minimum information required to maintain the Surface Contact Profile at the display system is as follows (for each contact):

TIME: 21:13:04 DATE: 06/04/79 CONTACT # H3 STATISTICS		OWNSHIP STATISTICS	
COURSE	: 315 DGS	COURSE	: 315 DGS
SPEED	: 30 KTS	SPEED	: 40 KTS
BEARING	: 022P DGS	LATITUDE	: 55.8N DGS
RANGE	: 073290 YDS	LONGITUDE	: 058.6W DGS
CPA TIME	: 00.00 H.M	QUADRANT NUMBER:	11
CPA DISTANCE	: 000000 YDS		
LATITUDE	: 51.3N DGS		
LONGITUDE	: 047.7W DGS		
COLLISION STATUS:			
QUADRANT NUMBER:	06		

CONTACT STATISTICS TO SCREEN.
ENTER CONTACT ID: F# OR H# OR U# . U1
IS INPUT DATA CORRECT? (Y/N) _

SPECIFIC CONTACT QUERY AND RESULT

FIGURE 13

TIME: 00:28:14		DATE: 00/00/00					
CONTACT	QUADRANT	STATUS	TYPE	COURSE	SPEED	BEARING	RANGE
F0	12	FR		315 DG	40 KTS	135S DG	009370YD
F1	11	FR		315 DG	40 KTS	000 DG	000000YD
F2	12	FR		315 DG	40 KTS	135S DG	018750YD
H0	06	HO		315 DG	30 KTS	018P DG	075880YD
H1	06	HO		315 DG	30 KTS	020P DG	082650YD
H2	06	HO		315 DG	30 KTS	024P DG	080280YD
H3	06	HO		315 DG	30 KTS	022P DG	073290YD
U0	09	UN		180 DG	20 KTS	048P DG	105700YD
U1	08	UN		334 DG	20 KTS	073S DG	051020YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD

OUTPUT GRAPHIC DISPLAY TO SWITCH (S)
 (0)LP ONLY (1)PLASMA.ONE (2)PLASMA.TWO (3)PLASMA.ONE AND PLASMA.TWO
 INPUT SWITCH NUMBER: 0,1,2 OR 3. _

HARDWARE REASSIGNMENT

FIGURE 14

- Latitude
- Longitude
- Course
- Speed
- x-coordinate of display
- y-coordinate of display
- Quadrant
- Range
- Bearing
- Collision status
- CPA Time
- CPA Distance

The display system will have the following functional capabilities:

- Remote processing must include capability to minimize time and length of data reception;
- Data reception must be on an interrupt basis in order not to circumvent the processing and formatting of data and the creation of local data bases;
- Graphical and statistical display modes to present the surface contact profile of the operational area;
- An adequate man-machine interface must be implemented to provide real-time (response) interaction with the system;
- The real-time (response) interaction must provide for a zoom-in capability for any contact of interest;
- Levels of correctness at the computer and operator levels must be achieved to minimize operator errors; and

- Hardware devices and the software displays must be driven independently to allow for maximum system flexibility and redundancy. This will allow the operation of the system in a degraded mode.

D. TECHNOLOGICAL BASE

Minicomputers, Micro-processors, Microcomputer and Plasma graphic display devices form the core of the technological base on which the system is based.

Minicomputers are designed as powerful computational tools. Among their major features are fast control processors with choices of semiconductor and core memory, floating point processing, sophisticated memory management schemes and an extended I/O capability.

Minicomputers such as the PDP-11/50 are designed for high speed real time applications and for large multi-user and multi-task applications. The PDP-11/50 Minicomputer, because of its computational and I/O capabilities, in conjunction with its availability at the Naval Postgraduate School was selected as the main frame computer of the system.

Microprocessors, in general, are not complete computers, but Central Processor Units (CPU) implemented with, say, one to ten large-scale-integrated-circuit chips. Large Scale Integration (LSI) chips are comprised of 1,000 or more gates; many LSI chips hold over 6,000 gates or a "Complete Central Processor."

The term "Micro-computer" is even less well defined. This is a small-stored program computer comprising memory

and input/output circuits together with a Microprocessor CPU.

Microcomputers with less than 2000 words of memory cost between \$25 and \$1,000 in quantities of 100. Practically all are special-purpose computers used as system components.

Interface design and programming make the mass-produced microcomputers and/or minicomputers into new special-purpose dedicated machines.

The Intellec Microcomputer Development System (MDS) is a complete, coordinated computer system, designed around Intel's 8080 microprocessor. The MDS has a 2 micro-second instruction cycle, a repertoire of 72 powerful instructions, unlimited subroutine nesting, and a versatile interrupt scheme.

The Microcomputer Development System (MDS) was selected as the host computer for the Display system because of its 8080 microprocessor, its memory capabilities, its interrupt mechanism, and its extended I/O capabilities.

The graphical display device selected was the 2500 Plasma-scope Gas Discharge System.

The plasma panel contains 262,144 individual dots which are capable of being discretely addressable in terms of selecting specific x and y coordinate values for excitation; e.g., to create or extinguish light. Plasma Panels do not require refresh and, once a particular point on the display is "turned-on", it continues to glow until "turned-off."

The CRT device selected for alphanumeric display of informational data and utilization as the Man-Machine Interface device was the DATAMEDIA ELITE 2500 Video Terminal. The DATAMEDIA is a stand-alone terminal containing an alphanumeric display, keyboard, storage, control logic and a asynchronous/synchronous communications interface.

The data link selected (2400 baud) for the system was an asynchronous serial/communication line based on EIA standard RS-232-C specifications. The link was selected because it was hardware compatible and was available at N.P.S.

The Intel's Microcomputer Development System (MDS) hosts the ISIS-II operating system, which supports 8080 assembly language and the PL/M-80 high-level language. Because of the complexity and size of the Software package anticipated, the PL-M/80 high level language was chosen for its multiple and powerful features.

The PDP-11/50 Minicomputer hosts the UNIX operating system which supports high level languages such as Fortran, Pascal and C. Since "C" is the system's language and it has the largest set of library functions, this language was chosen as the high level language to be used at the PDP-11/50. A more detailed description of the hardware components is given in Chapter IV and its associative appendices.

E. DEVELOPMENT PLAN

Upon completion of a 'top-down analysis' of the effort required to implement the system, the project was divided

into three major areas:

1. Hardware and Software functions and capabilities to be developed for the PDP-11/50 Minicomputer.
2. Hardware and Software functions and capabilities to be developed for the MDS Microcomputer.
3. Hardware and Software interfaces required to provide computer to computer communications between the PDP-11/50 Minicomputer and the MDS Microcomputer.

From the analysis, a critical development path was determined creating a development order (priority, time basis) for the three major areas of the project.

The most critical project area was the Software and Hardware interfaces for computer to computer communications. There did not exist any systems compatible interface (hardware or software) between an MDS Microcomputer and the PDP-11/50 Minicomputer. Further complicating the effort was the fact that the MDS microcomputer is an 8-bit word machine, hosting the ISIS-11 operating system supporting the PL/M-80 language, and the PDP-11/50 is a 16-bit machine hosting the UNIX operating system supporting the "C" language.

At completion of the interface, the effort was directed to the other two project areas in a Flip-Flop manner allowing interim testing of the partial system configuration. Greater emphasis was placed on the Hardware and Software development at the MDS end, because of the greater proportion of development effort required in this project area. The iterative Flip-Flop development was highly desirable to provide feedback

to the development process. The iterative process allowed the development of an adequate test data set to test and validate the system design upon completion.

The Project development was evaluated under the following guidelines:

- All Hardware components and interfaces were tested on a stand-alone basis before incorporating them into the system design being implemented.

- Software development was approached in such a manner as to allow testing at the procedure and module level.

- Upon completion of the System design implementation, the system was evaluated utilizing a test data set to demonstrate system capabilities compared against design objectives.

III. DISPLAY MODES

A. GENERAL INFORMATION

The design approach taken in the implementation of the display modes was a mix of complementary graphical and alphanumeric types of information.

The strategy was to present graphical data at the plasmascope, and alphanumeric data at the CRT screen or the line printer. Operator interaction was provided through the CRT keyboard.

The interaction was designed to provide the operator with the necessary capability to query the system, to obtain real-time general and individual contact statistics. This could be done with full system operation and in a degraded mode. The operator Command Options are given in table 1. For detailed description of the Command Options see appendix A.

In order to provide correctness in the interaction of the operator with the system, a method to check the operator's input was implemented,

B. COMMAND OPTIONS: CORRECTNESS

Each Command option has a built-in procedure whose function is to allow only correct inputs in the command. The parameters of each command are bounded within certain ranges that the operator must respect, if not, the following message will appear at the CRT screen:

COMMAND OPTIONS

B.....; DISPLAY GENERAL DATA STATISTICS TO CRT SCREEN.
G.....; DISPLAY GENERAL DATA STATISTICS TO LINE PRINTER.
S.....; DISPLAY GENERAL DATA STATISTICS PER DATA RECEPTION.
L-F#,L-H#,L-U#...; DISPLAY CONTACT STATISTICS TO SCREEN.
P-F#,P-H#,P-U#...; DISPLAY CONTACT STATISTICS TO LINE PRINTER.
W.....; DISPLAY CURRENT TIME AND DATE TO SCREEN.
T.....; RESET DATE AND TIME.
R.....; DATA RECEPTION VERIFICATION TO SCREEN.
Z.....; SET STATUS OF PLASMA (LAND2) AND LINE PRINTER.
M.....; DISPLAY COMMAND OPTIONS TO SCREEN.
N.....; DISPLAY COMMAND OPTIONS TO LINE PRINTER.

PLASMA TOUCH PANEL

TABLE 1

INPUT DATA INCORRECT: ENTER CORRECT DATA.

This message will remain in the CRT screen until the correct data is entered and only then the operator can continue with the command.

After a Command function is performed, the system will query the operator with the following message:

IS INPUT DATA CORRECT? (Y/N)

The operator has the option to complete the command by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the CRT screen reflecting the operator's choice. In case of a 'Y', the Command is executed, in case of a 'N', the Command state will be reinitialized automatically to request Command input parameters from the operator, disregarding his last input.

This process will continue until the input data is accepted by the system and by the user.

C. GRAPHICAL DISPLAY MODE

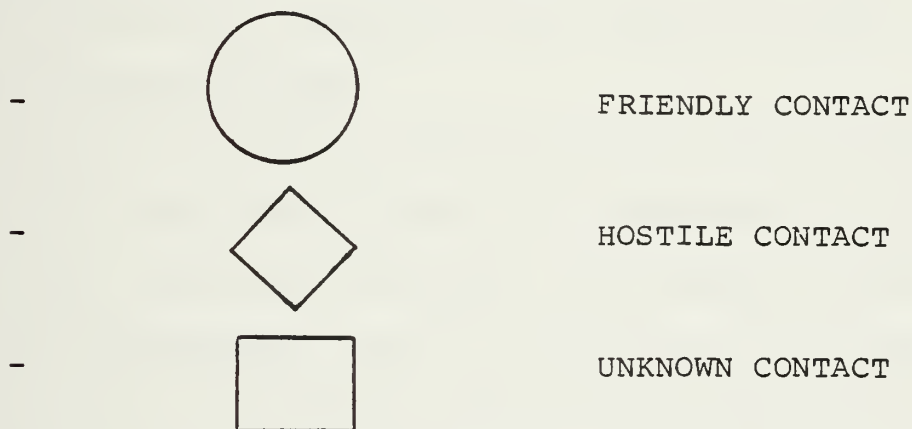
The graphical data presented at the primary plasma screen is a geographic region representing an absolute area 80 by 80 miles, partitioned in 16 numbered quadrants each 20 miles square, and symbolic representation of friendly, hostile and unknown contacts.

The geographic region represented in the screen is determined by the PDP-11/50 which has the responsibility for

establishing the base longitude and latitude of the tactical environment. The sixteen 20 mile square quadrants remain fixed (with respect to the screen) to provide for enhanced visual interpretation of the data and to provide a grid pattern for the user interaction with the touch panel device.

All of the above applies to the secondary plasma device, except for the touch-panel capability (no touch - panel device installed at the secondary plasma).

The symbolic representation of friendly, hostile and unknown contacts are as follows:



All the symbols had an adjacent numeric Id (0-9) representing the order of detection of the contact relative to its contact set. Each of the contact types; friendly, hostile and unknown defines a contact set.

The symbolic representation and the numeric ID of each contact set are displayed at the primary and secondary plasma devices to denote contact position within the geographic region.

The representations of contact speed and course is restricted to the primary plasma screen. The representation

of contact course is given by a vector-tail which originates at the center of the contact symbol, with its length determined by the contact speed (0 to 40 knots).

An illustration of a primary display is shown in Figure 8. An illustration of a secondary display is shown in Figure 10.

D. ALPHANUMERIC DISPLAY MODE

In order to be able to display concurrently the alphanumeric statistical data and allow communications of the operator with the system via the command options, the CRT screen was partitioned.

The 25 line screen was divided into two sections, the first 19 lines were allocated for alphanumeric data, and the last 6 lines were allocated for operator communications.

The alphanumeric data is presented at the screen in Table format, representing the General Data characteristics (all contacts), and a specific Contact characteristics also showing ownship characteristics.

An illustration of the General Data Statistics is shown in Figure 15.

An illustration of the Specific Contact Statistics is shown in Figure 16.

E. DATA ELEMENTS

The data elements used in the presentation of the alphanumeric data, once processed, produce a variety of information which defines the following relative to 'ownship':

- all Contacts
 - i.e. General Data characteristics
- A Specific Contact
 - i.e. Contact data characteristics.

Appendix B describes the data elements in detail.

ILLUSTRATION OF GENERAL CONTACT STATISTICS

TIME: 14:12:21 DATE: 05/09/79

CONTACT	QUADRANT	STATUS	TYPE	COURSE	SPEED	BEARING	RANGE
F0	16	FR		063 DG	20 KTS	027S DG	009370YD
F1	16	FR		063 DG	20 KTS	000 DG	000000YD
F2	16	FR		063 DG	20 KTS	027S DG	018750YD
H0	10	HO		045 DG	10 KTS	122P DG	076590YD
H1	10	HO		045 DG	10 KTS	125P DG	083160YD
H2	10	HO		045 DG	10 KTS	129P DG	080460YD
H3	10	HO		045 DG	10 KTS	127P DG	073650YD
U0	09	UN		180 DG	20 KTS	132P DG	122410YD
U1	08	UN		334 DG	20 KTS	038P DG	057580YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD
				DG	KTS	DG	YD

FIGURE 15

ILLUSTRATION OF SPECIFIC CONTACT STATISTICS

TIME: 14:12:21 DATE: 05/09/79

CONTACT # HL STATISTICS		OWNSHIP STATISTICS	
COURSE-----:	045 DGS	COURSE-----:	063 DGS
SPEED-----:	10 KTS	SPEED-----:	20 KTS
BEARING-----:	125P DGS	LATITUDE-----:	60.4N DGS
RANGE-----:	083160 YDS	LONGITUDE-----:	060.3W DGS
CPA TIME-----:	00.00 H.M	QUADRANT NUMBER---:	16
CPA DISTANCE-----:	000000 YDS		
LATITUDE-----:	54.2N DGS		
LONGITUDE-----:	048.5W DGS		
COLLISION STAUTS-----:			
QUADRANT NUMBER-----:	10		

FIGURE 16

IV. HARDWARE COMPONENTS

Given below is a list of the Hardware components used in this study and discussed in this chapter.

- 1 PDP-11/50 Minicomputer
- 1 MDS Microcomputer
- 2 2500 Plasma-Scope Gas Discharge Display System
- 1 Plasma-Scope Touch Panel
- 1 Datamedia Elite Video Terminal
- 1 Line Printer

A. MICROCOMPUTER DEVELOPMENT SYSTEM "MDS" DESCRIPTION

The Intellec microcomputer development system (MDS) is designed around Intel's popular 8080 microprocessor. The MDS utilizes the INTEL SYSTEMS IMPLEMENTATION SUPERVISOR (ISIS-II), as its operating system in conjunction with the INTELLEC system "Firmware Monitor" package.

The 8080 has a 2- μ sec instruction cycle, a repertoire of 72 powerful instructions, unlimited subroutine nesting, and a versatile interrupt scheme. The 8080 supports up to 65,536 (64K) words of memory and up to 512 I/O devices (256 input, 256 output). The basic hardware configuration includes 16,384 (16K) bytes of Random-Access-Memory (RAM), and six fully implemented I/O interfaces to:

- a Teletype (including its paper tape reader)
- a CRT terminal (or other compatible device),
- a high-speed paper tape reader,

- a high-speed paper tape punch,
- a line printer, and
- Intel's Universal PROM Programmer.

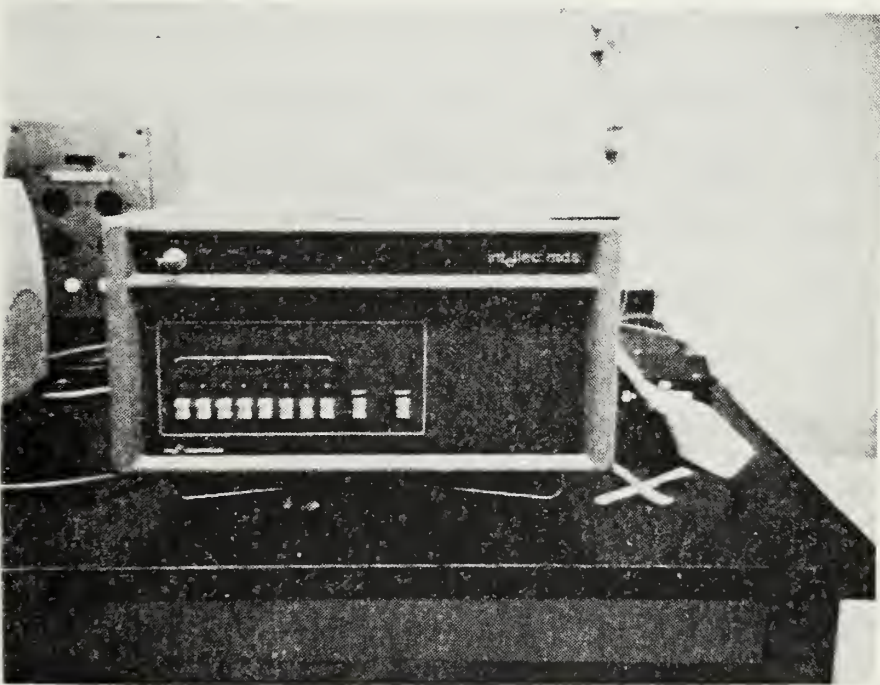
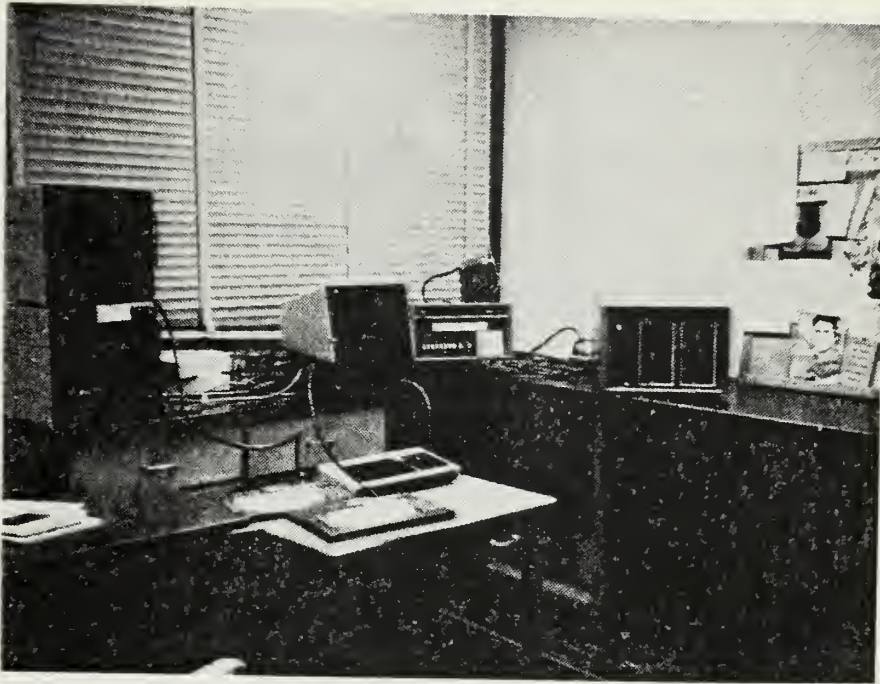
The "standard" configuration of the overall computer system consists of the MDS microcomputer, a dual-diskette drive (a quarter million bytes per floppy disk), a CRT and/or teletype for man-machine interface, a resident high-level PL/M-80 compiler, and a resident assembly language 8080/8085 macro assembler, as shown in figure 17.

B. PLASMA PANEL DESCRIPTION

This section describes the SAI Technology Company's Model 2500 Plasma-scope Gas Discharge Display System, which is capable of displaying alphanumeric characters and/or graphics. The plasmascope also features several configuration options to provide full capability for interfacing with a keyboard and various parallel and serial computer interfaces such as:

- The Interface I/O.
- The Display Buffer.
- The Vector Generator.
- The Character Generator.
- The Manual-Entry Keyboard.

The plasma panel contains 262,144 individual dots which are capable of being discretely addressable in terms of selecting specific x and y coordinate values for excitation; e.g., to create or extinguish light. The panel is normally driven by selecting parallel groups of lines on one axis (Y)



MDS MICROCOMPUTER CONFIGURATION

FIGURE 17

and scanning on the other (X). This operation provides displays of alphanumeric data using a dot matrix/format for characters and symbols. Also, by selecting single element location in coordinated fashion, graphics can be created on the display surface.

More specifically, the plasma panel consists of two panels of clear glass each of which has embedded parallel electrodes that are appropriately separated. The panels, aligned with the electrodes at 90 degrees, are separated by a dielectric and space seal as shown in picture 18. This spacer area is filled with a neon-based gas.

In use, each electrode is sequentially activated with an a.c. sustaining voltage. The amplitude of this voltage is controlled so that a breakdown, which causes localized ionization, occurs only at an intersection where the sustained voltage is augmented with an additional appropriate voltage. The ionized gas emits visible light and is maintained in this state by the sustained signal; thus, inherent memory exists. To erase this spot of light, a reverse polarity signal is placed on the electrode pair at the appropriate time; this results in a net reduction of the voltage below the sustaining level and the subsequent collapse of the ionization field.

The Model 2500 Plasmascope has the following interface capabilities:

- Parallel I/O Buffer, 16 bits.
- Differential I/O Buffer.

PLASMA SCOPE PANEL CONSTRUCTION

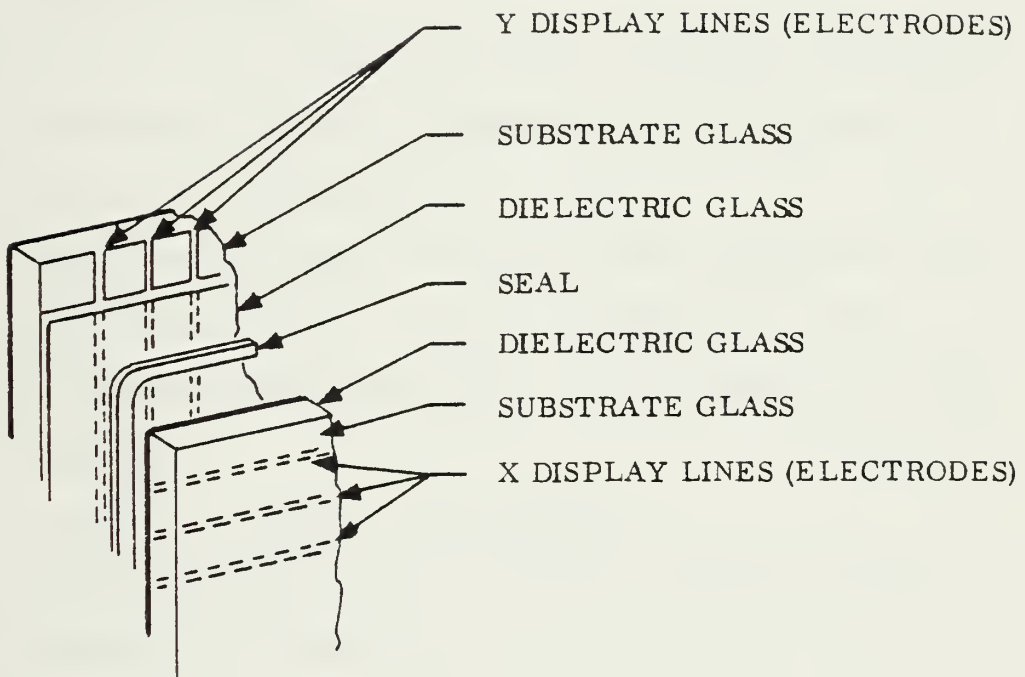


FIGURE 18

- Serial I/O Buffer.
- Data Channel Controls.
- Synchronous Interface.
- Asynchronous Interface.
- NTDS - 3V I/O

The Plasmascope has the following advantages over the conventional CRT display:

- No filaments or heaters.
- No external components such as magnetic deflection coils or yokes.
- No delicate precisely aligned internal components.
- Inherently digital, thus requires no digital to analog conversion.
- Much lower volume and weight when compared to an equivalent viewing area available in a CRT.
- Inherently stronger than a large evacuated glass envelope.
- No refresh memory requirements.
- No brightness variation between characters or elements of characters.
- No jitter or extraneous noise.

C. PLASMASCOPE TOUCH PANEL DESCRIPTION

The touch panel is an input device for Plasmascope which allows the operator to touch the display panel and input positional information to the computer. The touch panel uses a crossed array of light beams projected just above the

display surface. When an x and y beam is broken by an obstacle such as a finger, the panel inputs the x-y address to the computer. The panel then waits until the finger is repositioned before sending a new address.

There are 16 horizontal and vertical light beams that create a grid pattern of 256 positions which can be identified by the touch panel logic; this logic transforms this positional data to an eight bit data word.

The word consists of 4 bits representing the horizontal position and 4 bits representing the vertical position (x and y coordinates respectively, referenced to the left upper corner). These two 4-bit nibbles in conjunction with the touch-panel status bit, makes the positional information available to the user for any compatible I/O device.

D. DATAMEDIA ELITE 2500 TERMINAL DESCRIPTION

The DATAMEDIA Elite 2500 Video Terminal (shown in Figure 19) is a stand-alone terminal containing an alphanumeric display, keyboard, storage, control logic and a synchronous/asynchronous communications interface.

The Elite 2500 can receive at data rates from 50 to 9600 baud synchronous or asynchronous with a screen capacity of 1920 characters.

The Elite 2500 can store and identify 128 ASCII characters. The standard display format is 80 character line by 25 lines. Each character can be stored as a form field character or Blink field character, or both. All characters are formed on a 5 x 7 dot matrix.



DATAMEDIA ELITE 2500 VIDEO TERMINAL

FIGURE 19

The cursor, which is an underline, will identify the position on the screen. The next character received will be entered.

The DATAMEDIA Elite 2500 is an ideal CRT terminal because of its extended set of "control functions" and the following device attributes:

- quiet operation
- editing plus roll mode
- 50 to 9600 baud
- 80 characters per line
- no end of line hang ups
- protected field
- computer derived or high light field (blink)
- addressable cursor
- added carriage time with printer transmit
- good reliability
- electronic keyboard

E. PDP-11/50 MINICOMPUTER DESCRIPTION

The PDP-11/50 is a powerful 16 bit minicomputer of the PDP-11 family of PDP-11 processors, ranging from board microcomputers to full multipurpose computer systems.

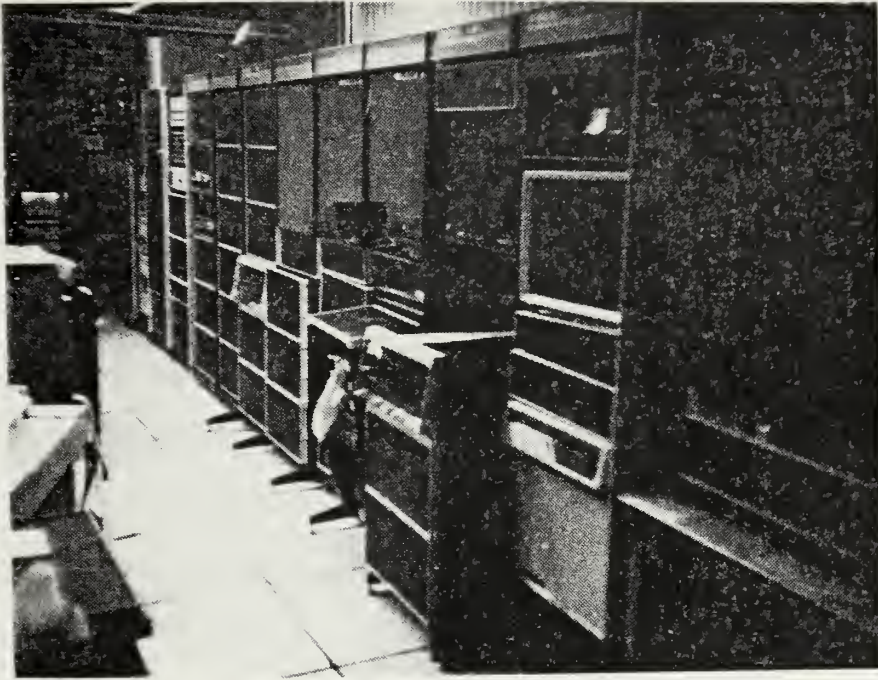
All of the processors are built upon a common architecture, that uses a similar instruction set and input/output systems. The programs developed on one PDP-11 processor may therefore run on any other PDP-11 processor without major conversions. Hence all PDP-11 machines are architecturally similar and hardware and software upwards compatible.

The PDP-11/50 is designed as a powerful computation tool for high-speed real-time applications and for large multi-user, multi-task applications, requiring up to 124 K words of addressable memory space. It will operate with solid state and core memories, and includes many features not normally associated with 16-bit computers. Among its major features are a fast central processor with choices of semiconductor and core memory, an advanced FLOATING POINT PROCESSOR, and a sophisticated memory management system.

The 11/50's basic I/O interface include the following:

- alphanumeric display
- teletype
- high-speed line printer
- card reader
- synchronous and asynchronous communications devices
- storage devices. Storage devices range from small reel magnetic tape units to mass storage magnetic tapes and disk memories. A large number of storage devices, in any combination, may be concurrently connected to the PDP-11/50 systems.

The configuration utilized in the MDS-PDP interface involves a PDP-11/50 system running under the UNIX operating system in a multi-user mode with the overall hardware system configuration as shown in figure 20.



PDP-11/50 MINICOMPUTER CONFIGURATION

FIGURE 20

V. MDS - PDP INTERFACE

A. GENERAL INFORMATION

The computer to computer interface involved the establishment of a communication/data link between the Intellec Microcomputer Development System (MDS) and the Digital Equipment Corporation (DEC) PDP-11/50 minicomputer system. The interface required asynchronous serial data/communication (based on EIA standard RS-232-C line) data link, and a Handler ("data" flow controller) at each respective computer. The handlers were required in order to convert input data to appropriate system format(s) for assimilation into the system(s) environment. The handling involved the initiation of a system response to manipulate the data set as required; i.e.,

- creation of a file
- updating of a file
- process commands
- systems calls
- initiation and termination of communications.

The interface provides the capability to edit, compile, and execute files in the PDP-11/50 minicomputer and in the MDS microcomputer. The interface provides for the transfer of files in three operational modes. The first two modes consist of a one way transfer of a file from either the MDS to the PDP-11/50 or the PDP-11/50 to the MDS. The third mode

of operation involves the "round-robin" transfer of a file initiated at either the MDS or PDP-11/50 to the other and after completion of interaction the original updated file is transferred back to the originator. The most significant operational aspect of this interface is that it provides the combined system development capabilities of both; the Intellec Microcomputer Development System (MDS) and the Digital Equipment Corporation (D.E.C.) PDP-11/50 minicomputer systems, to a user of either or both of the systems.

For a detailed description of the Data link, MDS hardware modifications and PDP software modifications see appendix C.

B. SOFTWARE COMPONENTS: FUNCTIONAL DESCRIPTION

1. MDSPDP PL/M-80 Program

MDSPDP operates by maintaining two buffers, one for the characters typed at the CRT, and the other for the characters sent to and from the MDS microcomputer. In each buffer, characters are arranged as a FIFO queue. The CRT buffer is 200 bytes long. The PDP buffer is 9 K bytes long, beginning at memory location D400 Hex and ending at the memory location F800 Hex which is the last memory location available for storage. This PDP buffer can be expanded up to 40K by changing the base of the buffer as needed. When the CRT buffer is full and the operator tries to type more characters, MDSPDP reacts by sending five (5) "Beep" characters to the MDS CRT. When the PDP buffer is full and the PDP

tries to send more characters, MDSPDP reacts by sending an error message to the MDS CRT, terminating receive state and reentering neutral state. If this is the case the PDP file must be partitioned within the bounds of the buffer.

MDSPDP uses the ISIS-II system calls to transmit byte data from the PDP and CRT buffers to the PDP-11/50, and, to and from the floppy disk to the MDS system. The ISIS-II operating system will issue general error messages concerning software and/or hardware faults to the MDS CRT.

The main routine in the MDSPDP handler is an infinite loop which performs a polling operation on the status bits of the TTY usart and the CRT usart; then depending on the system's state, appropriately inputs or outputs data to and from the CRT or PDP buffers.

The critical feature of the MDSPDP program is asynchronous timing of data communications, program execution, system control, and handshaking, between the PDP-11/50 Unix operating system and the MDS ISIS-II operating system. This was done while maintaining man-machine interfaces to the MDS system and to the PDP system through the MDS microcomputer.

2. PDPRECEIVE "C" Program

One of the two programs (user mode) written to work in conjunction with MDSPDP is the PDPRECEIVE program which is written in "C", and executes under the PDP-11/50 UNIX operating system. The functions of this program are to act as an input handler for byte data, creation of a file for

the input data, and perform the "handshaking" with the "MDSPDP" program executing under the ISIS-II operating system on the MDS microcomputer.

3. PDPSEND "C" Program

The other program used (user mode) in conjunction with the MDSPDP is the PDPSEND program, which is also written in "C" and executes under the PDP-11/50 UNIX operating system. The function of this program is to act as an output handler for the PDP-11/50 transmitting byte data from an existing file in the PDP to the MDS microcomputer.

C. INTERFACE DEFINITION

The MDS-PDP-11 data communication link is based on the following hardware and software components.

1. PDP-11/50

a) Single input/output port (CRT) to the PDP-11/50 configuration.

b) Software input and output handlers for PDP-11/50: PDPRECEIVE and PDPSEND programs respectively, both programs written in high level language "C" supported by the UNIX operating system.

2. DATA LINE

a) ASYNCHRONOUS 2400 baud serial data/communication line (based on EIA standard RS-232-C line) from the PDP-11/50 system to the Intellec MDS system.

3. MDS

a) CRT input/output port at the MDS system as shown in figure 21. (Local CRT interface to MDS via teletype (TTY) input/output port.)

MDS DATA LINK I/O CONFIGURATION

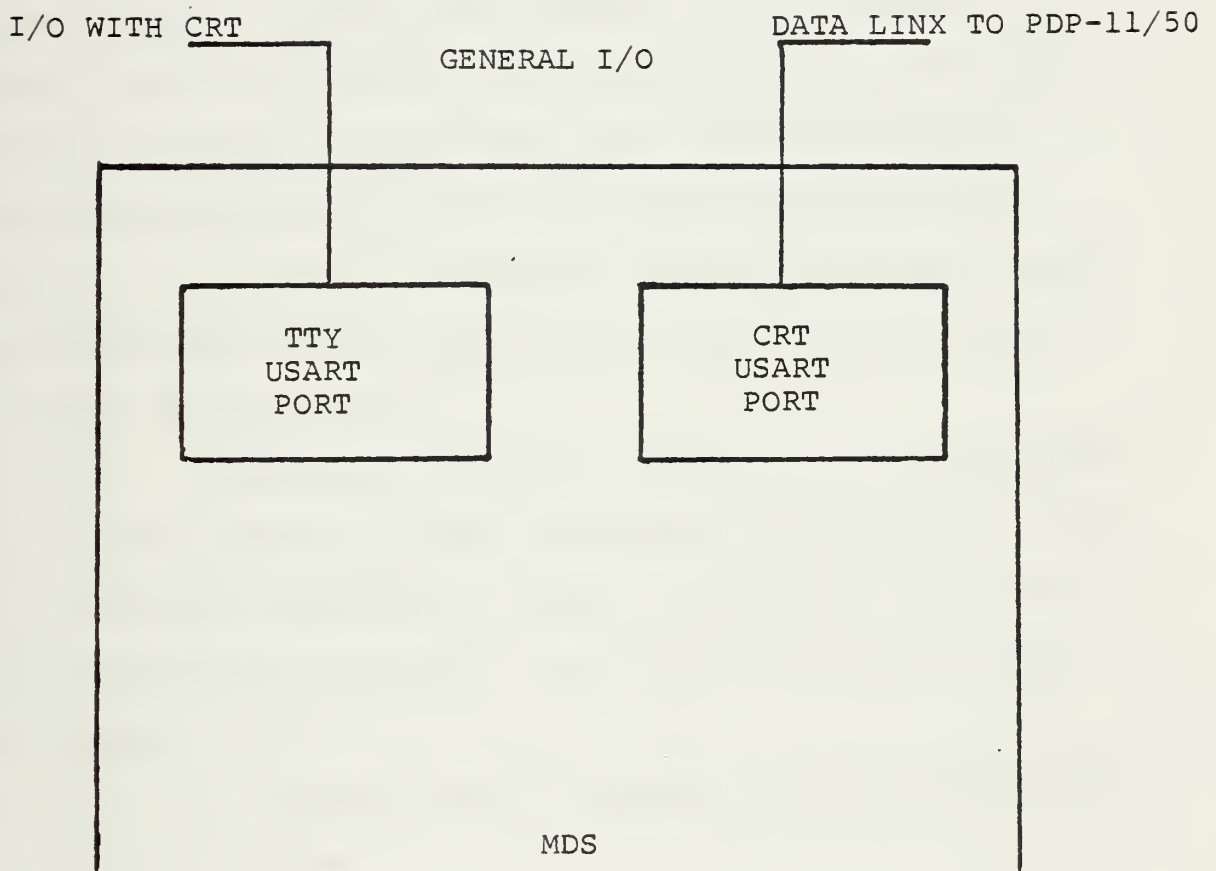


FIGURE 21

b) Software input/output handler for the MDS-PDP data link: MDSPDP program written in high-level language PL/M-80 supported by the ISIS operating system.

In order to establish communications between the MDS microcomputer and the PDP minicomputers; the MDSPDP handler, "MDSPDP," must be executed at the MDS microcomputer system. The purpose of this is to create the necessary local environment (system parameters, system interfaces, input/output buffers, asynchronous timing, etc.) in order to present the MDS SYSTEM as a CRT terminal to the PDP-11/50 operating system, via the data line. This provides an operator at the MDS system, with log-on capability and access to all system functions of the UNIX operating system (program execution and termination, creation, deletion, updating of files, etc.).

Given that communications have been established, the MDSPDP handler acts as a software switch allowing three (3) different communication modes of operation, and a program termination capability. The state of operation are as follows:

- Netural state: CRT(MDS) to PDP, PDP to CRT (MDS)
- Receive State: PDP to floppy disk via MDS microcomputer.
- Transmit state: floppy disk to PDP via MDS microcomputer.

The Netural state is automatically entered with the execution of the "MDSPDP" handler at the MDS microcomputer. It is in this state that the operator at the MDS system is provided with log-on capability and access to the UNIX operating system.

Once the log-in has been completed, the system will still be in the neutral state, awaiting operator commands to enter receive state, transmit state, stay in neutral state or terminate program.

The receive state provides the MDS system with a data import capability to receive data from the PDP. The MDS microcomputer has the system's responsibility to receive data, store it in core until completion of the data transfer (End-of-file) and, then transfer the data in core to floppy disk and return to netural state.

The transmit state provides the MDS system with a data export capability to transmit data from the MDS system to the PDP. The microcomputer has the system's responsibility to latch (transfer) 128 byte blocks of data from the floppy disk to core, transmit it from core to the PDP, and repeating the process until end-of-file, and returning to neutral state. This is accomplished by the PDPRECEIVE and the MDSPDP programs.

D. PDP-11/50-MDS INTERRUPT DRIVEN DATA TRANSMISSION

In order to create the necessary environment for the display system, concurrent with the transmission and

reception of files between the PDP-11/50 MINI computer and the MDS MICRO computer, interrupt driven data communications and interrupt driven processes were implemented.

The interrupt mechanisms consisted of the occurrence of an internal operating system event such as the completion of a process, the status of the real time system's clock, and the readiness of the PDP-11/50 to communicate data to the MDS microcomputer.

The interrupt driven mechanisms allow the MDS to operate in a remote stand alone basis, while maintaining the capability to respond to "external" events with complete data reception capability. For detailed information about the interrupt driven method see appendix C.

VI. SOFTWARE OVERVIEW

A. GENERAL INFORMATION

The software application package for the System utilized 2 different high level languages; "PL/M-80" in the MDS Microcomputer, and "C" in the PDP-11/50 Minicomputer [Ref. 9,10,11,12,13,14,15,16,17,20,22 and 25].

The software effort for the MDS microcomputer required 70 percent (%) of the project development time for completion. The Software package consisted of 24 major modules, 217,000 bytes of source code and 39,000 bytes of object code.

The Software effort for the PDP minicomputer required 15 percent (%) of the project development time. This Software package consisted of 13 subroutines, 10,000 bytes of source code and 12,000 thousand bytes of object code.

The software development effort consisted of top-down design and bottom-up testing [Ref. 6,7 and 8].

B. DATA STRUCTURES

To establish compatible data bases at each computer, the same data structure format was utilized for both machines. This was done to facilitate the creation and formatting of data at the PDP-11/50 for transmission and reception and building of the data bases at the MDS end. The "structure" statement in PL/M-80 and C was used because of its inherent attributes for building data bases. For additional information

on this powerful statement in each language see references 13, 25. See figure 22 for data structures illustration.

C. M.D.S. DISPLAY SYSTEM SOFTWARE

The development of the software package for the M.D.S. microcomputer evolved the software into functional modules. These modules consist of stand-alone procedures to provide for testing at the procedure and module level. A functional description of each one of the modules used, with its internal subroutines is presented in appendix D.

D. PDP-11/50 SOFTWARE

As stated previously, the integration and interface between the software development packages at both ends, in conjunction with the MDS and PDP-11/50 hardware, provides the functional capabilities of the system. The programming effort at this end was directed to the creation of an algorithm using the facilities of the UNIX operating system. This algorithm provided the appropriate environment and Data Base to input and output 'raw data' through the data link to the MDS system.

A functional description of each procedure is presented in appendix D.

"PL/M-80" DATA STRUCTURE

```
DECLARE SHIP$PLOT (15) STRUCTURE (  
    LAT(10) ADDRESS,  
    LONG(10) ADDRESS,  
    COURSE(10) ADDRESS,  
    SPEED(10) BYTE,  
    X$BOW(10) ADDRESS,  
    Y$BOW(10) ADDRESS,  
    QUADRANT(10) BYTE,  
    RANGE(10) ADDRESS,  
    BEARING(10) ADDRESS,  
    COLLISION$FLAG(10) BYTE,  
    CPA$TIME(10) ADDRESS,  
    CPA$DISTANCE(10) ADDRESS,  
    COUNT BYTE ) PUBLIC ;
```

"C" DATA STRUCTURE

```
struct {  
    int lat[10] ;  
    int longtd[10] ;  
    int course [10] ;  
    char speed[10] ;  
    int xbow[10] ;  
    int ybow[10] ;  
    char quadrant[10] ;  
    int range[10] ;  
    int bearing[10] ;  
    char collflag[10] ;  
    int cpatime[10] ;  
    int cpadist[10] ;  
    char count;  
} shipplot[3], *shipptr;
```

FIGURE 22

VII. SYSTEM DEVELOPMENT SUMMARY

The following discussion presents the development of the shipboard tactical-situation display system. The first section describes the development of the Interface capability between the MDS Microcomputer and the PDP-11/50 Minicomputer on a stand-alone basis providing an integrated MDS-PDP Software and Hardware development facility. The second section describes the entire system development effort incorporating the interface described in section A.

A. COMPUTER TO COMPUTER INTERFACE

The computer to computer interface implemented involved the establishment of a communication data link between the MDS and the PDP, utilizing I/O drivers at each end.

The first step taken in the establishment of this interface was to select and physically connect a data line.

Due to the availability of only serial I/O capability compatible to both computers, a 2400 baud EIA standard RS-232-C line was selected. The second step taken was to create the I/O Software drivers, to receive, send and manipulate data to complete the interface.

Because of the different operating systems and application languages at each computer; distinct and language incompatible I/O drives had to be written at each computer. The incompatibility between I/O hardware utilized at each computer, further complicated by word size (8 bit word and 16 bit word

at the MDS and the PDP respectively), created a necessity for extensive manipulation of data. Temporary buffers and extensive computer to computer handshaking by the I/O drivers at each computer were necessary to allow for the creation, transmission and reception of files.

The data manipulation and handshaking required local to the MDS I/O driver was such that the execution time only permitted a maximum data reception baud rate of 2400, rather than the 9600 baud rate initially attempted.

B. HARDWARE AND SOFTWARE COMPONENTS

This section describes the development of the Hardware and Software components implemented at the MDS and PDP-11/50 computers, and their integration into the system.

1. MDS Microcomputer

a) The initial configuration of the system consisted of a single plasma display interfaced with the MDS micro-computer, and a software package implemented to drive the plasma device to display graphics and alphanumerics. This Hardware and Software interface was done by Babin and Seaman, in NPS thesis "A Microcomputer Based Plasma Display System" in March 1978 [Ref. 26].

Taking the existing configuration and its capabilities, a Software package was initiated to support the graphical requirements of the Shipboard Tactical-situation display system.

Parallel to this effort, a Hardware interface of a second plasma display was initiated by expanding the I/O

capability of the MDS Microcomputer compatible with the first plasma device already installed. Upon completion of the Hardware interface of the second plasma device to the MDS Microcomputer, a second Software interface was implemented to drive the second plasma display. This was done to provide the same graphical and alphanumeric capability as the original plasma device. At this point the Software package initiated to support the graphical requirements of the system was concluded and tested. The software package was then expanded by including the necessary functions to support both plasma displays with different functional capabilities in the same Software package.

The system configuration, consisting of the MDS microcomputer, the two plasma devices along with their hardware and software interfaces, and the graphical package for the system were tested on a stand alone basis to evaluate and validate the graphical capabilities for later integration into the system being developed.

b) With this configuration complete, the next step was to implement the Data management scheme in charge of receiving the raw data sent by the PDP-11/50 (over the MDS-PDP interface), placement in memory (Temporary buffers, with known delimiters), and from there relocation into preformatted Data Structures.

These Data Structures are the Data Bases local to the MDS Microcomputer. The Data is extracted, interpreted, formatted and then handled by the graphical display package to present the informational data at the plasma devices.

To test the Data management scheme, the graphical display package, interface line and handlers were integrated into the hardware already implemented. Then a test data-set was transmitted from the PDP over the interface to the MDS microcomputer to be handled by the data management system and presented as graphical information at both plasma devices.

c) At this point of the development, the basic system framework has been implemented and tested. Data has been successfully transmitted from the PDP-11/50 to the MDS, where it was processed, handled and presented as graphical information.

Due to the limited information that could be presented in a graphical display format in a clear, and unambiguous manner, the need for a complete alphanumeric presentation was obvious. Although an alphanumeric display capability was defined in the early design, an extended capability was implemented because of the graphic display limitations. The alphanumeric display software package consists of two alphanumeric tables created and formatted to present statistical contact information describing the operational environment. One of the tables consists of information about all the contacts in the operational environment. The other table consists of information about a specific contact and information about a specific ownship. The alphanumeric display software package was first tested on a stand-alone basis, then integrated into the current

framework of the system. The new configuration was tested and evaluated, for consistency and validation of both types of display formats (graphical display and alphanumeric display).

d) At this stage in the project development, all major hardware and software components had been implemented providing the basic system configuration to receive data, process data, handle data and display graphical and alpha-numerical information.

The system, as configured, did not provide any interactive facility to allow the operator access to a specific piece of information via any display mode. To eliminate this problem a Man-Machine Interface software package, system compatible, was developed and integrated into the system.

Special attention was paid to the speed of response in the operators-System interaction to avoid psychological step down. The following general design principles were taken into account in the design and implementation of the Man-Machine Interface:

- Simple interface with user: Interaction is accomplished by pressing a single key at the CRT keyboard for each command option. This type of interaction makes the display system very easy to use.

- Self-explanatory: Each command option implemented in the display system provides the operator with the appropriate instructions to execute the command. These

instructions are written to the CRT screen to guide the operator through the command.

- Interaction by anticipation: The display system is constantly prompting the operator at the CRT screen. As a result of the prompting, the operator can identify at any time the current status of the display system.

- Speed of response: Immediate response and feedback to the operator are provided in order to minimize operator distraction.

- Provide choice of methods: A mix of display modes, as explained in chapter three, and command options provide the operator with the capability to select the type of display.

A special feature incorporated in the design of the Man-Machine interface was the integration of a light emitting diode Touch-Panel to one of the Plasma devices. The hardware and software implementation of the Touch-Panel allowed the operator the opportunity to interact directly with the graphical display of informational data.

With the inclusion of the Man-Machine-Interface into the System configuration, the Shipboard Tactical-Situation System at the MDS level was fully implemented and extensively tested.

2. PDP-11/50 Minicomputer

The system development at the PDP-11/50 Minicomputer consisted of a software application package that provided the following capabilities:

a) Command and Control of the Shipbaord Tactical-Situation System including the real time system's clock.

b) Generation of test data (surface contact profile) to simulate operational scenario to test and evaluate display system operation.

c) Capability to abstract from raw data the necessary contact parameters to create the information data bases to transmit to the MDS Microcomputer via the MDS-PDP interface.

Upon completion of the Software application package, the Shipboard Tactical-Situation System (including all Hardware and Software components at the PDP-11/50, MDS and data link) was tested and evaluated utilizing the capability of the PDP-11/50 to generate test data.

VIII. CONCLUSIONS AND RECOMMENDATIONS

The minicomputer based interactive display system implemented in this study is a workable design for the

- Access and retrieval of graphical and alphanumerical information on request.
- Control of the flow of information to the different display devices.
- Accomplishment of the level of correctness expected at the operator and systems level.
- Remote processing required for the reception, handling and formatting of data.

The integration of the capabilities of the display system in conjunction with the computer to computer interface allows the emulation of a Shipboard Tactical situation.

The most critical technical aspect of the project was the implementation of the serial computer to computer interface. The design of the interface required an extensive knowledge of both computers input/output capabilities and operating systems. The information data bases at the 16-bit word PDP minicomputer had to be broken down to bit patterns, transmitted to the 8-bit word MDS microcomputer and then from the bit patterns, reconstructed into the MDS information data bases.

The display system was enhanced by including two plasma scopes to the system. The two plasma scopes not only provided

a back-up capability but also provided for the interaction between the plasma scopes. The interaction provided by the plasma touch panel allowed the operator direct access to graphical data presented. This capability greatly increased the operator's ability to evaluate the tactical situation in a real time manner.

The hardcopy feature available in the system was valuable because it provided redundancy which increases the flexibility and reliability of the system. And, also provided hardcopy historical back up in case of system degradation caused by a loss of a display device, i.e., plasma scope, CRT screen.

It is recommended that to improve the system performance, study should be made in the following areas:

- Expand the alphanumeric hardcopy capability to include graphical hardcopy capability. This will provide a more complete and accurate hardcopy historical back-up.
- Consider the replacement of the DATAMEDIA CRT and Keyboard with a plasma scope. This will permit the utilization of a more powerful display device and it will increase the reliability by eliminating the only cathode-ray-tube device in the system.
- The installation of touch panel devices at all plasma scopes. This will be specially important in the plasma scope used for the interaction with the operator. This is because it will substantially

eliminate the need for a keyboard by replacing the keyboard with software driven "menus" presented at the screen.

- Consider the replacement of the MDS 8-bit machine by a 16-bit microcomputer. This will increase the speed of execution. With this change, the speed of transmission of data should increase, which is highly desireable.

APPENDIX A
OPERATORS MANUAL
FOR THE
DISPLAY SYSTEM

This operator's manual describes the Man-Machine interface of the "DISPLAY SYSTEM" at the Naval Postgraduate School Computer Laboratory.

NAME:

DISPLAY GENERAL DATA STATISTICS TO CRT SCREEN.

DESCRIPTION:

This command loads the most recent data set received and time of request into the General data statistics format to be presented at the CRT screen.

FORMAT:

'B' Enter the command state and the following message
 will appear at the CRT screen:

GENERAL DATA STATISTICS TO SCREEN (Y/N)

The operator has the option to complete the command by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the CRT screen reflecting the operator's choice.

NAME:

DISPLAY GENERAL DATA STATISTICS TO LINE PRINTER.

DESCRIPTION:

This command loads the most recent data set received and time of request into the General Data statistics format to be printed at the Line Printer.

FORMAT:

'G' Enter command state and the following message will appear at the CRT screen:

GENERAL DATA STATISTICS TO PRINTER (Y/N)

The operator has the option to complete the command by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the CRT screen reflecting the operator's choice.

NAME:

DISPLAY GENERAL DATA STATISTICS PER DATA RECEPTION

DESCRIPTION:

This command sets a software switch to either load the data set just received at the MDS microcomputer and time of reception into the General Data Statistics format to be presented at the CRT screen immediately after the Data reception, or bypass this option.

FORMAT:

'S' Enter the command state and the following message
 will appear at the CRT screen:

PRINT GENERAL DATA STATISTICS PER DATA RECEPTION (Y/N)

The operator has the option to set the switch by inputting a 'Y' or to bypass the option by inputting a 'N'.

A 'YES' or a 'NO' will be printed to the MDS CRT screen reflecting the operator's choice.

NAME:

DISPLAY CONTACT STATISTICS TO SCREEN.

DESCRIPTION:

This Command loads the most recent user specified contact data set and time of request into the Contact statistics format to be presented at the CRT screen.

FORMAT:

'L' Enter the command state and the following message will appear at the CRT screen:

CONTACT STATISTICS TO SCREEN (Y/N)

The operator has the option to continue in the command state by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the screen reflecting the operator's choice.

If the operator continues in the command state the following messages will appear at the CRT screen:

CONTACT STATISTICS TO SCREEN.

ENTER CONTACT ID:F# or H# or U#.

Command Option

At this point in the command state, the operator may select any contact that is in the current General Data Statistics. The user 'must' select an allowable two digit alphanumeric contact designator to complete the command.

NAME:

CONTACT STATISTICS TO LINE PRINTER.

DESCRIPTION:

This command loads the most recent user specified contact data set and time of request into the contact statistics format to be printed at the Line Printer.

FORMAT:

'P' Enter the command state and the following message will appear at the CRT screen:

CONTACT STATISTICS TO LINE PRINTER (Y/N)

The operator has the option to continue in the command state by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the screen reflecting the operator's choice.

If the operator continues in the command state the following message will appear in the CRT screen:

CONTACT STATISTICS TO LINE PRINTER.

ENTER CONTACT ID: F# or H# or U#.

Command Option

At this point in the Command state the operator can select any contact that is in the current General Data statistics. The user 'must' select an allowable two digit alphanumeric contact designator to complete the command.

NAME:

CURRENT TIME AND DATE TO SCREEN.

DESCRIPTION:

This command presents the current Date and Time to the CRT screen.

FORMAT:

'W' Execute command and the current date and time message will appear at the CRT screen in the following format:

TIME: XX:XX:XX DATE XX/XX/XX

- Time is given in hours, minutes and seconds.
- Date is given in Month, day and year.

NAME:

RESET DATE AND TIME.

DESCRIPTION:

This command allows the user to reset the Date and Time.

FORMAT:

'T' Enter the command state and the following message
will appear at the CRT screen:

ENTER NEW DATE/TIME (Y/N)

The operator has the option to continue in the command state by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or 'NO' will be printed to the CRT screen reflecting the operator's choice.

If the operator continues in the command state the following messages will appear at the CRT screen:

INPUT DATE AND TIME.

YEAR:

The operator may enter the 2 digit year. The year entered will be echoed to the right of 'YEAR:', upon acceptance of the year, 'MONTH:' will be appended to the same line as year. This interaction will be the same for 'MONTH:', 'DAY:', 'HOURS:', 'MINUTES:' and 'SECONDS:'.

Example:

YEAR: 79 MONTH; 05 DAY: 31 HOURS: 14 MINUTES: 00 SECONDS: 00

NAME:

DATA RECEPTION VERIFICATION TO SCREEN.

DESCRIPTION:

This command sets a software switch to either acknowledge received state and print the number of bytes transmitted from the PDP-11/50 minicomputer to the MDS microcomputer per data reception to the CRT screen, or bypass this option.

FORMAT:

'R' Enter the command state and the following message will appear at the CRT screen:

VERIFY DATA RECEPTION. (Y/N)

The operator has the option to set the switch by inputting a 'Y' or to bypass the option by inputting a 'N'. A 'YES' or a 'NO' will be printed to the CRT screen reflecting the operator's choice.

Example:

During the reception of data the following message will appear at the CRT screen:

SYSTEM IN RECEIVE STATE:

After the reception of the updated data the following messages will be added to the previous one at the CRT screen:

PDP PROMPTING: END OF RECEPTION.

PDP BUFFER/FILE WRITTEN TO STRUCTURE.

CHARACTER COUNT: XX BYTES TRANSMITTED FROM PDP TO STRUCTURE.

NAME:

SET STATUS OF PLASMA (1 and 2) AND LINE PRINTER

DESCRIPTION:

This command sets software switches to select the primary display device per data reception for the presentation (device dependent) of graphical or statistical data to the plasma displays or the line printer respectively.

The options are as follows:

- (0) Line printer only.
 - General Data Statistics per data reception to line printer only.
- (1) Plasma One.
 - Graphic display of General Data Set.
- (2) Plasma Two.
 - Graphic display of General Data Set.
- (3) Plasma ONE and Plasma TWO.
 - Plasma ONE is the primary Graphic display device (Provides capability for touch panel command option), with plasma TWO as a back-up.

FORMAT:

'Z' Enter the command state and the following message will appear at the CRT screen:

REASSIGN PLASMA DEVICES (Y/N)

The operator has the option to continue in the command state by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed to the CRT screen reflecting the operator's choice.

If the operator continues in the command state the following messages will appear at the CRT screen:

OUTPUT GRAPHIC DISPLAY TO SWITCH (#).

(0) LP ONLY (1) PLASMA ONE (2) PLASMA TWO (3) PLASMA ONE
and PLASMA TWO.

INPUT SWITCH NUMBER: 0, 1, 2 OR 3.

At this point in the command the operator must select an allowable single digit switch number to complete the command.

NAME:

COMMAND OPTIONS TO SCREEN

DESCRIPTION:

This command lists the command options at the CRT screen. The Commands listing will remain at the CRT screen during the current and the next data reception.

FORMAT:

'M' Enter command state and the following message will appear at the CRT screen:

LIST COMMAND OPTIONS: (Y/N)

The operator has the option to complete the command by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed at the CRT screen reflecting the operator's choice.

NAME:

COMMAND OPTIONS TO THE LINE PRINTER

DESCRIPTION:

This command prints the Command options at the line printer.

FORMAT:

'N' Enter command state and the following message will appear at the CRT screen:

LIST COMMAND OPTIONS TO LINE PRINTER (Y/N)

The operator has the option to complete the command by inputting a 'Y' or exiting the command state with no action being taken by inputting a 'N'. A 'YES' or a 'NO' will be printed at the CRT screen reflecting the operator's choice.

TOUCH PANEL:

The Touch Panel as described in Chapter III has been implemented in the System as a media (when both plasma devices are active) to draw the contacts in any quadrant and their positions relative to the 'ownship' -- a dashed line is also drawn from the contacts to the 'ownship'.

In order to perform this Command Option the operator must touch any quadrant of the grid draw in Plasma One and the result as explained in the previous paragraph will appear in Plasma Two and will remain until the next updated data is received, unless, the operator decides to touch another quadrant and another picture will appear in Plasma Two.

If the operator performs this command option when the display system is in receive state, the system will bypass the option, and after the receive state is finished only the 'ownship' will appear in Plasma Two, at this point the operator is clear to perform the touch panel option again.

APPENDIX B
DATA ELEMENTS DESCRIPTION

This appendix describes the data elements for the alphanumeric display of the "DISPLAY SYSTEM" at the Naval Postgraduate School Computer Laboratory.

DESCRIPTOR:

CONTACT

FORMAT:

AN-----Where A stands for an alphanumeric digit, N
stands for a numeric digit.

RANGE:

A.- F, H or U representing friendly, hostile or
unknown respectively.

N.- 0,1,...,9.

DESCRIPTION:

These 2 digits represent a specific contact ID, which
is a reference name for display purposes.

DESCRIPTOR:

QUADRANT

FORMAT:

NN----- Represents a quadrant identified in the grid
drawn at the primary and secondary plasma
screens.

RANGE:

NN 0,1,...,16

DESCRIPTION:

Is a positional reference point for display purposes,
and user interaction with the touch panel device.

DESCRIPTOR:

STATUS

FORMAT:

AA----- Represent 2 alphanumeric characters,
indicating if the contact is friendly, hostile
or unknown.

RANGE:

- FR Meaning Friendly Contact.
- HO Meaning Hostile Contact.
- UN Meaning Unknown Contact.

DESCRIPTION:

The characters represent the classification of a
specific contact.

DESCRIPTOR:

TYPE

FORMAT:

AAA----- Represent 3 alphanumeric digits, indicating
the class of the contact.

RANGE:

As required.

DESCRIPTION:

To be implemented in the future, depending on the
System's user. This Descriptor is associated with
classified information.

DESCRIPTOR:

COURSE

FORMAT:

III----- Indicates Contact direction relative to the
absolute North, measured in Degrees (DG.)

RANGE:

000 Degrees to 359 Degrees.

DESCRIPTION:

Defines Contact course at the primary plasma display,
and at the alphanumeric display at the CRT screen and the
line printer.

DESCRIPTOR:

SPEED

FORMAT:

II----- Indicates the real velocity at sea of a given
contact, measured in Knots (KTS).

RANGE:

00 Knots to 99 Knots.

DESCRIPTION:

Define Contact speed at the primary plasma display, and
in the alphanumeric display at the CRT screen and the line
printer.

DESCRIPTOR:

BEARING

FORMAT:

IIIP or

IIIS----- Where III is the numeric value of the bearing
and S/P indicates starboard or port
respectively.

RANGE:

000 Degrees

000 Degrees to 179 P

000 Degrees to 179 S

180 Degrees

DESCRIPTION:

Defines the angle between the ownship bow and the
position of a given contact.

All bearings are relative to 'ownship'.

DESCRIPTOR:

"RANGE"

FORMAT:

IIIIII----- Where IIIIII is the distance between 'ownship'
and a given contact, measured in yards (YD).

RANGE:

0 yards to 226,275 yards.

DESCRIPTION:

Indicates the distance between the 'ownship' and any
given contact within the limits of the 80 mile square
geographic region. The distance is measured in yards.

DESCRIPTOR:

LATITUDE

FORMAT:

II.IS or

II.IN----- Where II.I is the geographic position of any
contact in the South Hemisphere or North
Hemisphere. Measured in degrees and tenths
of degree.

RANGE:

00.0 degrees to 89.9 degrees North

or

00.0 degrees to 89.9 degrees South.

DESCRIPTION:

Is a geographical positional reference for any contact,
used in conjunction with longitude to give an exact
position in the geographic region.

DESCRIPTOR:

LONGITUDE

FORMAT:

III.IW or

III.IE----- Where III.I is the geographic position of any
contact in the West or in the East. Measured
in degrees and tenths of degree.

RANGE:

000.0 degrees to 179.9 W

or

000.0 degrees to 179.9 E.

DESCRIPTION:

Is a geographical positional reference for any contact,
used in conjunction with latitude to give an exact position
in the geographic region.

DESCRIPTOR:

CPA TIME

FORMAT:

HH.MM----- Given in hours (HH)
and minutes (MM)

RANGE:

00.00 to 24.00

DESCRIPTION:

The Closest point of approximation is the time in which any specific contact is going to be at the nearest possible point to the 'ownship' if and only if the ship courses of both are known, and at least two bearings to the contact are taken in a lapse of 3 minutes.

If the CPA time is negative or greater than 24, is automatically set to zero.

DESCRIPTOR:

CPA DISTANCE

FORMAT:

IIIIII----- Where IIIIII is the distance between the
 'ownship' and any contact at CPA time.
 Measured in yards (YDS).

RANGE:

0 yards to 160,000 yards

DESCRIPTION:

The distance between any given contact and the 'ownship'
at CPA TIME is called CPA DISTANCE.

DESCRIPTION:

COLLISION STATUS

FORMAT:

flashing message:

"COLLISION"

RANGE:

non applicable

DESCRIPTION:

When a collision status (course) with a given contact is identified; this status is presented to the operator in the appropriate format (see format).

DESCRIPTOR:

TIME

FORMAT:

HH:MM:SS---- Where HH represents hours,
Where MM represents minutes,
Where SS represents seconds.

RANGE:

Hours. - 00 to 23,
Minutes. - 00 to 59,
Seconds. - 00 to 59.

DESCRIPTION:

The System maintains a real time clock at the MDS end;
Time is utilized in all the statistical Displays and also
is available on request by the operator.

DESCRIPTOR:

DATE

FORMAT:

MM/DD/YY---- Where MM represents month,
Where DD represents day,
Where YY represents year.

RANGE:

MM. - 1 to 12,
DD. - 1 to 31,
YY. - 00 to 99.

DESCRIPTION:

The System accepts the date as an input, and increments the day accordingly through the current month, requiring resetting at the beginning of each month. The date is available on request by the operator.

APPENDIX C
MDS-PDP INTERFACE
DETAILED DESCRIPTION

A. DATA/LINK CABLE-CONNECTOR DESCRIPTION

The data link used was an asynchronous serial/communication line based on EIA standard RS-232-C specifications [Ref. 35].

The data line was connected to the PDP-11/50 distribution bus on the DZ-11 I/O card via a DB25S socket.

The data line was connected to the MDS CRT I/O port via a DB25P connector/plug.

The respective pin connections for the data line socket and connector/plug are given in Figure 23.

B. MDS HARDWARE MODIFICATIONS

The Standard Configuration was modified as follows in order to establish the communication/data link between the Inteltec Microcomputer development system (MDS) and the PDP-11/50.

MDS modifications:

The asynchronous serial data/communication link, based on an EIA standard RS-232-C line, was connected to the CRT input/output port. The local MDS CRT device had to be configured to operate connected to the TTY input/output port with a current loop 20 ma. line. The

DATA LINK PIN CONNECTIONS

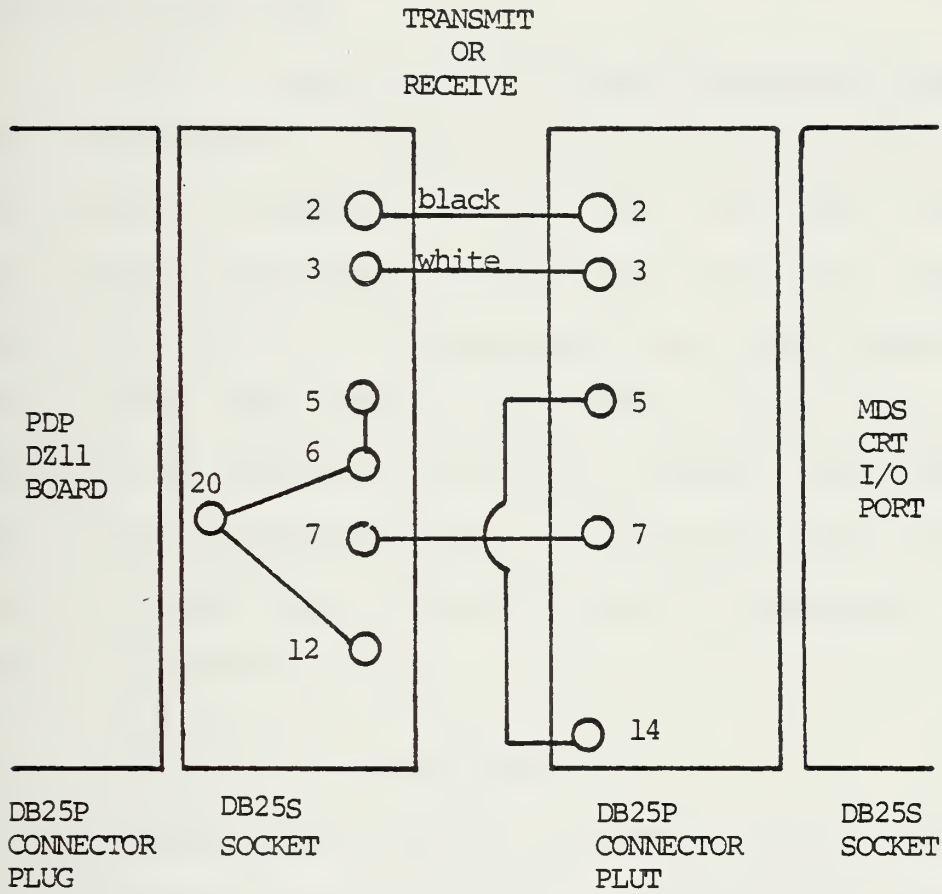


FIGURE 23

"bootstrapping" action was accomplished at this CRT device.

The baud rate for the CRT input/output port usart was set at 2400 baud by connecting jumpers 19-20 in baud rate jumper pad network to allow data transmission to and from the PDP-11/50.

Due to the length of the MDS Interface program, the MDS was incapable of receiving the correct information at 9600 baud. It should be noted that concurrent with setting the CRT usart baud rate to 2400, the TTY usart baud rate was set at 9600. It is imperative that the MDS CRT device and the TTY input/output usart are baud rate compatible. This was accomplished by setting (hardware switches) the CRT device to 9600 baud rate, and/or by setting the CRT device to 2400 baud rate and software programming the TTY usart to 2400 baud rate.

C. PDP SOFTWARE MODIFICATIONS

The asynchronous serial data/communication link, based on EIA standard RS-232-C line, was connected to the distribution bus of the DZ-11 I/O card (8-channel RS-232 I/O).

In order that the PDP-11/50 I/O be compatible with the MDS System I/O parameters, the following parameters were set in system software in the Unix operating system at the PDP 11/50 via the NGETTY.C program module. The NGETTY.C program module of the Unix operating system has the responsibility for setting line parameters for all PDP-11/50 I/O ports, i.e., baud rate, parity, etc.

DECLARATIONS OF NGETTY.C

```
#define HUPCL 01
#define XTABS 02
#define LCASE 04      // convert upper-case to lower
#define ECHO 010      // turn on echo
#define CRMOD 020
#define RAW 040
#define ODDP 0100
#define EVENP 0200    // set even parity
#define ANYP 0300     // set any parity
#define LEDIT 01      // set for line edit
#define NOERASE 01000 // no BS or ERASE capability
                        // (TTY, for instance)
```

```
/*
 * Delay algorithms
 */
```

```
#define CR1 010000
#define CR2 020000
#define CR3 030000
#define NL1 000400
#define NL2 001000
#define NL3 001400
#define TAB1 0020000
#define TAB2 0040000
#define TAB3 0060000
```

```
/*
 * speeds
 */
```

```
#define B110 3
#define B150 5
#define B300 7
#define B1200 9
#define B2400 11
#define B9600 13
```

```
#define SIGINT 2
#define SIGQTT 3
```

```
TABLE CONSTRUCT FOR MDS COMMUNICATIONS WITH THE
PDP-11/50 at 2400 BAUD (NGETTY.C)
/* table '6' -- 2400 */
'6', '6',
ANYP+RAW+NL1+CR1, ANYP+XTABS+ECHO+CRMOD+FF1,
B2400, B2400, LEDIT+NOERASE,
" n f 007login: ",
```


D. MDS INTERRUPT SYSTEM

The actual implementation of the interrupt system in the MDS Microcomputer involves the use of the PL/M-80 high level language constructs (i.e., interrupt "typed" procedure), the interrupt processing available in the PL/M-80 compiler resident in the ISIS-II operating system, and the interrupt logic (I/O device based interrupts) available on the MDS Hardware through the Monitor [Refs. 20,21,22,23,24, and 25].

The Monitor Interrupt logic groups the seven interrupt lines from the I/O devices; TTY, CRT, Paper tape reader, punch, and line printer interfaces into a status word that can be read, under program control, by the CPU. In addition, the interrupt logic, will, if enabled, issue an interrupt request on level 3 (levels 0,1,2,3,4,5,6,7 available) under the ISIS-II operating system when any one of the interface interrupt lines goes true.

The Monitor interrupt logic consists of six 8097 bus drivers, one 8093 bus driver, one 747D-type flip-flop, and assorted gating circuits as shown on sheet three of the module schematic, Figure 5-18 in Reference 24.

The design at the MDS system level required that the MDS recognize the occurrence of an interrupt from the PDP-11/50 uniquely at the Monitor Interrupt logic. This was accomplished by modifying the A-52 eight to one NAND gate (7430 IC) logic module to a single stage inverter.

This permitted only the recognition of an interrupt (from the PDP-11/50) at the CRT USART.

This capability permitted the MDS to be interrupt driven by the PDP-11/50, by recognizing the interrupt of the PDP, which is in charge of sending the updating data through the interface program.

E. PDP-11/50 INTERRUPT SYSTEM

The interrupt system implemented in the PDP-11/50 minicomputer is based on the recognition of the occurrence of a time interval (generated by the real time system's clock) and an interrupt structure built upon this timing. The interrupt system initiates an interrupt signal to the MDS system, coordinates and executes system to system handshaking to transmit file(s) to the MDS data bases.

APPENDIX D

SOFTWARE FUNCTIONAL DESSCRIPTION

A. MDS "PL/M-80" MODULES

A functional description of each one of the modules used is presented in this section. The subroutines contained in each module are also listed.

NOTE: A subroutine may appear in more than one module.

For a definition of "module" see reference 25.

This is the main program module, that initializes system parameters and interrupt mechanisms to maintain a system ready state. The system ready state enables data reception, data storage, data manipulation, and user interaction via the Command Options and Touch Panel.

This module is responsible for accessing the appropriate software module in order to provide graphical and statistical data display.

MODULE SUBROUTINES:

OUTPUT \$ STATUS \$ LP
SEND \$ CHAR \$ LP
PCRLF
SEND \$ STRING \$ LP
INITIALIZE \$ SCREEN
INITIALIZE \$ PRINTER
CLOSE \$ PRINTER
LIST \$ MENU \$ COMMANDS
LIST \$ MENU \$ COMMANDS \$ LP
ORIG \$ STRUCTURE \$ WRITE
NEW \$ STRUCTURE \$ WRITE
ORIG \$ PDP \$ WRITE
MAKE \$ ADDRESS
LOCATE \$ STRUCTURE \$ PDP
INT \$ CHECK
INT \$ 3
INT \$ 6

MAIN PROGRAM SOURCE CODE.

This module provides for the operator initiation of time and date, maintains the correct time through the utilization of the MDS real time clock in interrupt one, provides the capability of loading the current time and date in the appropriate data structure.

MODULE SUBROUTINES:

- INITIALIZE \$ MO \$ ARRAY
- CLOCK
- LOAD \$ TIME
- ERASE \$ LINE
- CHECK \$ YES \$ NO
- CHECK \$ INPUT
- BACK \$ SPACE \$ ERASE
- INITIATE \$ TIME

This Module declares ISIS-II operating procedures utilized in the manipulation of system files.

PROCEDURE CALLS:

OPEN

CLOSE

READ

WRITE

EXIT

CONSOL

DELETE

ERROR

RENAME

This Module enables and disables specific TTY baud rates, establishes split screen capability, and performs the function of a 'driver' for the input and output of characters and strings to and from the CRT.

MODULE SUBROUTINES:

```
SET $ TTY $ 9600
SET $ TTY $ 2400
OFF $ CRT $ KEYBOARD
ON $ CRT $ KEYBOARD
OUTPUT $ STATUS $ CRT
SEND $ CHAR $ CRT
CRLF
SEND $ STRING $ CRT
PRINT $ TO $ CRT
INPUT $ STATUS $ PDP
READ $ CHAR $ PDP
INPUT $ STATUS $ CRT
READ $ CHAR $ CRT
SET $ LOW $ HOME
CLEAR $ LOW $ SCREEN
CLEAR $ NEXT $ LINES
GET $ BYTE
CLOCK $ TIMER
```


This Module defines the appropriate formats in order to display the General Data Statistics table and the Specific Contact Statistics to the upper part of the CRT screen.

MODULE SUBROUTINES:

```
WRITE $ BIG $ PICTURE
WRITE $ LITTLE $ PICTURE
CONVERT $ ADDRESS $ TO $ CHARS
CONVERT $ BYTE $ TO $ CHARS
LOAD $ DATA $ BIG $ PICTURE
LOAD $ DATA $ LITTLE $ PICTURE
LOAD $ LINE $ ARRAY
BLANK
LOAD $ BLANKS
INITIALIZE $ LOAD $ PICTURE
SET $ STRUCTURE $ ZEROES
```


This Module writes the General Data Statistics and the Specific Contact Statistics to the line printer.

MODULE SUBROUTINES:

OUTPUT \$ STATUS \$ LP

SEND \$ CHAR \$ LP

LPCRLF

WRITE \$ BIG \$ PICTURE \$ LP

WRITE \$ LITTLE \$ PICTURE \$ LP

This Module declares the externals procedures previously declared in SCREEN.MOD and WRITE.LP to be included in MODULES requiring access to these procedures.

MODULE SUBROUTINES:

CONVERT \$ ADDRESS \$ TO \$ CHARS
CONVERT \$ BYTE \$ TO \$ CHARS
WRITE \$ BIG PICTURE \$ LP
WRITE \$ BIG PICTURE
WRITE \$ LITTLE PICTURE \$ LP
WRITE \$ LITTLE PICTURE
LOAD \$ DATA \$ LITTLE PICTURE
INITIALIZE \$ LITTLE PICTURE
SET \$ STRUCTURE \$ ZEROES

This Module clears and initializes both plasma screens, erases the grids and numbers the quadrants. It has the capability to draw solid and dashed contact symbols in plasma device one, and solid contact symbols in plasma device two. It also interprets the speed and course of the contacts in order to draw the appropriate tail. Maintains the 5 most recent data sets as back-up and is responsible for labeling (number ID) the contact symbols.

MODULE SUBROUTINES:

INITIALIZE \$ GRAPHICS

SEND \$ GRID \$ PLASMA

SEND \$ GRID \$ PLASMA \$ 2

INITIALIZE \$ STRUCTURE

DRAW \$ SHIP \$ DASH

DRAW \$ SHIP \$ DASH \$ LOOP

STRUCTURE \$ BACK

BACKUPS

DRAW \$ SHIP \$ PROCEDURE

This Module contains the same type of procedures as in GRAPH1-MOD, but they are directed to draw contact symbols in plasma device two.

MODULE SUBROUTINES:

DRAW \$ SHIP \$ DASH \$ 2

DRAW \$ SHIP \$ DASH \$ LOOP \$ 2

DRAW \$ SHIP \$ 2

This Module services the interrupt 6 produced by the touch panel device installed at plasma device one. After the interrupt is produced by touching the panel, a quadrant number is returned, which identifies which contact symbols (solid) are to be drawn at plasma device two. The ownship ship is also drawn at plasma device two and a dashed vector is drawn from every contact to the 'ownship'.

MODULE SUBROUTINES:

SERVICE \$ SIX

QUAD \$ NUM

This Module contains the External declarations for the Public procedures declared in GARCH1.MOD, GRAPH2.SRC and SERVE.SIX.

MODULE SUBROUTINES:

QUAD \$ NUM
INITIALIZE \$ GRAPHICS
INITIALIZE \$ STRUCTURE
SEND \$ GRID \$ PLASMA
SEND \$ GRID \$ PLASMA \$ 2
DRAW \$ SHIP \$ DASH
DRAW \$ SHIP \$ DASH \$ LOOP
STRUCTURE \$ BACK
BACKUPS
DRAW \$ SHIPS
SERVICE \$ SIX
DRAW \$ SHIP \$ DASH \$ 2
DRAW \$ SHIP \$ DASH \$ LOOP \$ 2
DRAW \$ SHIP \$ 2

This Module is responsible for the interrupt driven communications/interface between the MDS microcomputer and the PDP-11/50 minicomputer, when given control by the main module.

Coordinates 'handshaking', maintains temporary files and buffers to establish the Data link necessary to enable reception of the incoming Data Set produced by the PDP-11/50.

This Module consists of two submodules; MP.DEC which contains the declarations for PDP.MOD, and MP.COD which contains the main body of PDP.MOD.

1. MP.DEC

SYSTEM CALLS

OUTPUT \$ STATUS \$ CRT

SET \$ TTY \$ 2400

OUTPUT \$ STATUS \$ PDP

INPUT \$ STATUS \$ CRT

INPUT \$ CHAR \$ CRT

SEND \$ CHAR \$ PDP

CRLF

SEND \$ STRING \$ CRT

PRINT \$ TO \$ CRT

READ \$ CHAR \$ CRT

READ \$ CHAR \$ PDP

GET \$ CHAR \$ CRT \$ BUF

GET \$ CHAR \$ PDP \$ BUF
PUT \$ CHAR \$ CRT \$ BUF
PUT \$ CHAR \$ PDP \$ BUF
CRT \$ BUF \$ FULL
PDP \$ BUF \$ FULL
PRINT \$ HEX \$ NUMBER
FORMAT \$ HEX
PRINT \$ CHAR \$ COUNT
INT \$ CHAR \$ COUNT
COUNT \$ CHAR
INT \$ NEUTRAL \$ STATE
INT \$ RECEIVE \$ STATE
INT \$ TRANSMIT \$ STATE
BREAK \$ STATE
END \$ R
END \$ T
WRITE \$ RECORD \$ TO \$ DISK
WRITE \$ PDP \$ BUFFER
REBOTT

2. MP.COD

MAIN BODY OF PDP.MOD

This Module contains the System Level Plasma primitives which call the hardware plasma primitives utilized to create Display formats, grid patterns, and Symbolic representation of data. This Module consists of the union of six other Modules:

PLAEXT . ONE

PLAEXT . TWO

PLAPUB . ONE

UNKNOWN . SR1

PLAPUB . TWO

UNKNOWN . SR 2

This Module contains the System Procedures to:

- create and label grids
- create Friendly and Hostile solid and dashed symbols
- create ID number for each contact
- provide the erase capability for Friendly and Hostile symbols

for Plasma Device one.

MODULE SUBROUTINES:

WRITE \$ CONTACT \$ ID

DRAW \$ GRID

DRAW \$ FRIEND \$ SYMBOL

DRAW \$ FRIEND \$ DASH

ERASE \$ FRIEND \$ DASH

ERASE \$ FRIEND \$ SYMBOL

DRAW \$ HOSTILE \$ SYMBOL

DRAW \$ HOSTILE \$ DASH

ERASE \$ HOSTILE \$ SYMBOL

ERASE \$ HOSTILE \$ DASH

This Module draws unknown solid and dashed contact symbols at Plasma device one.

MODULE SUBROUTINES:

DRAW \$ UNKNOWN \$ SYMBOL

ERASE \$ UNKNOWN \$ SYMBOL

DRAW \$ UNKNOWN \$ DASH

ERASE \$ UNKNOWN \$ DASH

This Module contains the System Procedures to:

- create and label grid
- create Friendly and Hostile solid and dashed symbols
- create ID number for each contact
- Provide the erase capability for Friendly and Hostile symbols

for Plasma Device Two.

MODULE SUBROUTINES:

```
WRITE $ CONTACT $ ID $ 2
DRAW $ GRID $ 2
DRAW $ FRIEND $ SYMBOL $ 2
DRAW $ FRIEND $ DASH $ 2
ERASE $ FRIEND $ DASH $ 2
ERASE $ FRIEND $ SYMBOL $ 2
DRAW $ HOSTILE $ SYMBOL $ 2
DRAW $ HOSTILE $ DASH $ 2
ERASE $ HOSTILE $ SYMBOL $ 2
ERASE $ HOSTILE $ DASH $ 2
```


This Module draws unknown solid and dashed contact symbols at Plasma device two.

MODULE SUBROUTINES:

DRAW \$ UNKNOWN \$ SYMBOL \$ 2

ERASE \$ UNKNOWN \$ SYMBOL \$ 2

DRAW \$ UNKNOWN \$ DASH \$ 2

ERASE \$ UNKNOWN \$ DASH \$ 2

This Module contains the External declarations for the Public Procedures in PSPRIM.MOD, for Plasma primitives for Device one.

MODULE SUBROUTINES:

```
SET $ STATUS $ PLASMA
PLASMA $ WRITE
CLEAR $ PLASMA
PLASMA $ WRITE $ VECTOR
PLASMA $ PRINT $ STRING
INITIALIZE $ PLASMA
SET $ VECTOR
START $ VECTOR $ SOLID
STOP $ VECTOR $ SOLID
START $ VECTOR $ DASH
STOP $ VECTOR $ DASH
GRAPHIC $ DESIG
START $ ERASE $ VECTOR
STOP $ ERASE $ VECTOR
START $ ERASE $ DASH
STOP $ ERASE $ DASH
```


This Module contains the External declarations for the Public Procedures in PSPR2.SRC for plasma primitives for Device two.

MODULE SUBROUTINES:

```
SET $ STATUS $ PLASMA $ 2
PLASMA $ WRITE $ 2
CLEAR $ PLASMA $ 2
PLASMA $ WRITE $ VECTOR $ 2
PLASMA $ PRINT $ STRING $ 2
INITIALIZE $ PLASMA $ 2
SET $ VECTOR $ 2
START $ VECTOR $ SOLID $ 2
STOP $ VECTOR $ SOLID $ 2
START $ VECTOR $ DASH $ 2
STOP $ VECTOR $ DASH $ 2
GRAPIC $ DESIGN $ 2
START $ ERASE $ VECTOR $ 2
STOP $ ERASE $ VECTOR $ 2
START $ ERASE $ DASH $ 2
STOP $ ERASE $ DASH $ 2
```


This Module contains the external declarations for the public procedures of PLASMA.MOD, specifically the Sub-Modules of PLASMA-MOD.

- PLAPUB.ONE
- PLAPUB.TWO
- UNKNOW.SR1
- UNKNOW.SR2

MODULE SUBROUTINES:

```
WRITE $ CONTACT $ ID
DRAW $ GRID
DRAW $ FRIEND $ SYMBOL
DRAW $ FRIEND $ DASH
ERASE $ FRIEND $ DASH
ERASE $ FRIEND $ SYMBOL
DRAW $ HOSTILE $ SYMBOL
DRAW $ HOSTILE $ DASH
ERASE $ HOSTILE $ SYMBOL
ERASE $ HOSTILE $ DASH
DRAW $ UNKNOWN $ SYMBOL
ERASE $ UNKNOWN $ SYMBOL
DRAW $ UNKNOWN $ DASH
ERASE $ UNKNOWN $ DASH
WRITE $ CONTACT $ ID $ 2
DRAW $ GRID $ 2
```


DRAW \$ FRIEND \$ SYMBOL \$ 2
DRAW \$ FRIEND \$ DASH \$ 2
ERASE \$ FRIEND \$ SYMBOL \$ 2
DRAW \$ HOSTILE \$ SYMBOL \$ 2
DRAW \$ HOSTILE \$ DASH \$ 2
ERASE \$ FRIEND \$ DASH \$ 2
ERASE \$ HOSTILE \$ SYMBOL \$ 2
ERASE \$ HOSTILE \$ DASH \$ 2
DRAW \$ UNKNOWN \$ SYMBOL \$ 2
ERASE \$ UNKNOWN \$ SYMBOL \$ 2
DRAW \$ UNKNOWN \$ DASH \$ 2
ERASE \$ UNKNOWN \$ DASH \$ 2

This Module contains the Software primitives to handle the interface between the MDS microcomputer and the S.A.I. plasma device one, and the implementation of the following functions in software:

MODULE SUBROUTINES:

SET \$ STATUS \$ PLASMA
PLASMA \$ WRITE
CLEAR \$ PLASMA
PLASMA \$ WRITE \$ VECTOR
PLASMA \$ PRINT \$ STRING
INITIALIZE \$ PLASMA
SET \$ VECTOR
SET \$ VECTOR \$ ONE
START \$ VECTOR \$ SOLID
STOP \$ VECTOR \$ SOLID
START \$ VECTOR \$ DASH
STOP \$ VECTOR \$ DASH
GRAPHIC \$ DESIGN
START \$ ERASE \$ VECTOR
STOP \$ ERASE \$ VECTOR
START \$ ERASE \$ DASH
STOP \$ ERASE \$ DASH

This Module contains the Software primitives to handle the interface between the MDS microcomputer and the S.A.I. plasma device two, and the implementation of the following functions in software:

MODULE SUBROUTINES:

```
SET $ STATUS $ PLASMA $ 2
PLASMA $ WRITE $ 2
CLEAR $ PLASMA $ 2
PLASMA $ WRITE $ VECTOR $ 2
PLASMA $ PRINT $ STRING $ 2
INITIALIZE $ PLASMA $ 2
SET $ VECTOR $ 2
SET $ VECTOR $ ONE $ 2
START $ VECTOR $ SOLID $ 2
STOP $ VECTOR $ SOLID $ 2
START $ VECTOR $ DASH $ 2
STOP $ VECTOR $ DASH $ 2
GRAPHIC $ DESIGN $ 2
START $ ERASE $ VECTOR $ 2
STOP $ ERASE $ VECTOR $ 2
START $ ERASE $ DASH $ 2
STOP $ ERASE $ DASH $ 2
```


B. PDP "C" PROCEDURES

A functional description of each procedure is presented in this section.

This is the main program procedure that is responsible for the following:

- Initialization of system Parameters
- Creation of the appropriate Data Structures
- Provides the logic for the Interrupt mechanism
- Controls the timing between interrupts and the internal calls of the other Subroutines
- Provides the logic for the creation, manipulation and transmission of the Data to be sent to the MDS-end.

INITSTRU

This procedure is responsible for simulating an operational scenario, by generating a coherent data set over time relative to a tactical situation.

This procedure is a "Demo" program which generates the data that in real world will be generated by sensors interfaced with the main frame computer.

This procedure is invoked by the MAIN procedure each time an updated data set is required.

This procedure initializes the data structures to zero.

This initialization to zero of all data structures is necessary to clear the memory locations of the data structures.

This procedure is invoked by the MAIN procedure at the beginning of MAIN prior to the System Control Logic.

This procedure formats all 16-bit word data via bit manipulations prior to interacting with the output buffer. This formatting consists of transforming 16-bit words to two 8-bit words, respecting the bit hierarchy (relative position).

When these two 8 bit words are received by the MDS microcomputer, a 16 bit variable is created in memory by placing the lower and upper 8-bit words at the appropriate memory locations. The manipulation described above is necessary because the PDP-11/50 is a 16-bit machine vice the MDS, which is an 8-bit machine.

This procedure contains the appropriate data elements to close the transmission state at the PDP, provides delimiters to the data sets, and terminates the receiving state at the MDS-end.

This routine calls the procedures to calculate the data elements to be loaded into the data structures prior to transmission of the updated data set to the MDS-end.

The procedures called and respective functions are as follows:

1. LDLAT:

Calculates the latitude of a specific contact and loads the value to the data structure.

2. LDLONG:

Calculates the longitude of a specific contact and loads the value to the data structure.

3. LDCOUR:

Calculates the course of a specific Contact and loads the value to the data structure.

4. QUANDNUM:

Calculates the quadrant number (of the grid pattern) of a specific contact and loads the value to the data structure.

5. LDRANG:

Calculates the distance between any contact and the designated "ownship."

6. LDBEAR:

Calculates the bearing between any contact and the designated "ownship."

7. LDCPATD:

Calculates the CPA-TIME and CPA-DISTANCE between any contact and the designated "ownship."

APPENDIX E

DESCRIPTION OF GRAPHIC DISPLAY TECHNOLOGY

A. GENERAL INFORMATION

Most graphic display systems use refresh or storage technology. Three main types of refresh technologies exist: stroke writing, raster scanning and scan converting. Stroke writing display systems position an electron beam on the tube face much as one would draw on paper with a pencil. In raster scanning systems, the beam sequentially traces the entire face of the tube. When the beam arrives at a point that belongs to the picture under construction, a video signal brightens the beam to illuminate the screen. Hybrid scan converters use a storage tube to store the image and then scan the storage tube information onto a raster scanning monitor to display the image. Since the persistence of the phosphor in the tube is low, CRT's using one of these technologies require periodic image refreshing to prevent annoying screen flicker. These CRT's refresh the image at least forty times each second.

Two storage technologies exist: the storage tube and the plasma panel. With the storage tube, the CRT receives its image in the same way as a stroke writing system. However, the storage tube stores the image on a grid, eliminating periodic refresh. Unlike other graphic display systems, plasma panels do not use CRT's. The display consists of a

series of bright data that can be formatted into alpha-numerics and graphics. Plasma panels do not require refresh and, once a particular print on the display is "turned on," it continues to glow until "turned off."

B. PLASMA PANEL PHYSICAL DESCRIPTION

1. PANEL PARAMETERS

- a. Actual area; 8.55" x 8.55"
- b. Panel Glass size: 12.25" x 12.25"
- c. Light Spot Size: 10 to 12 mils
- d. Addressable Matrix: 512 x 512
- e. Dot spacing 0.0167" center-to-center
60 per inch
- f. Brightness: 50-foot Lamberts
- g. Contrast ratio: 25 to 1 (nominal)
- h. Color: Neon-range (585.2 manometers)

2. CHARACTER SIZE (other sizes dependent upon driving logic and/or software)

- a. 5 x 7 matrix 80 x 120 mils
- b. 7 x 9 matrix 120 x 150 mils

3. ELECTRIC DESCRIPTIONS

The Plasmascope primary power requirements are:

- a. 115 v ac
- b. 47 to 440 HZ
- c. 300 watts maximum
- d. single phase

4. PERFORMANCE

- a. Data Rates: addressable to the individual dot data at 50 KHZ;

b. Parallel Mode: 330 msec to address the entire screen. The parallel mode of operation allows the simultaneous addressing of 16 points in the Y axis. Y_0 , Y_8 and Y_4 through Y_7 (Y_0 , Y_8) of the Y address are used to select one of 32 sectors, each of which comprises 16 consecutive horizontal electrodes. The X address selects one column of 16 points in the addressed sector. The parallel address inputs are then used to address any number of the 16 points in the selected sector column.

5. DATA CODE

ASCII character set for alphanumeric operation (7 bit).

6. CHARACTER MATRIX

5 x 7 or 7 x 9 dot matrices.

7. RELIABILITY

The mean-time-between failure (MTBF) for the plasmascope is over 6,000 hours with JAN-TX parts at 25°C.

8. MAINTAINABILITY

The mean-time-to-repair (MTTR) for the plasmascope is 2 hours, board level maintenance or on-board replacement.

9. ELECTROMAGNETIC INTERFERENCE/TEMPEST

The plasmascope has been tested to MIL-STD-461 for EMI suppression. Several aspects of the model 2500 design are more critical for Tempest than for EMC. These include the display panel, the keyboard, and the power line. The display panel contains grid wires, approximately 8 inches

long, that contain signals which correlate with the information being displayed. A periphery coated metallic film on a face plate is provided for the display panel to minimize radiation. This precaution has been sufficient to permit other displays of similar design to meet the Tempest requirements.

The keyboard is somewhat exposed to radiation as discussed above for EMC. Because the exciting signal levels are small (HALL Effect voltages) and because the radiating elements are electrically very short at the processing signal frequencies, the keyboard is expected to satisfy Tempest requirements.

The power level conductor requirements of Tempest are met by a combination of filtering and consideration of Tempest requirements in the design of the power supply.

The display electronics operates at 50 KHZ, a frequency at which the EMI filters provide attenuation. The data lines between the Model 2500 and any data source can be designed for Tempest by providing adequate cable shielding to control EM radiation.

APPENDIX F

INTERRUPT STRUCTURES

A. BACKGROUND INFORMATION

A very large percentage of microprocessor systems are employed in control applications, i.e., situations in which the CPU controls a process in the "real" world by analyzing information concerning the behavior of the process.

Information is a measure of "surprise," and in many systems this equates as much to "when" as to "how much." To find out "when" an event occurs, it is possible to input and test status bits that are set by its occurrence. For a number of systems this may require almost continuous sampling while only a relatively few samples return much information. The machine can do no useful work while sampling and, thus, is inefficiently utilized. By allowing the events of interest to signal for the machines' "attention," the efficiency can be vastly improved. Thus, interrupt structures allow "event-driven" systems in which the concept of temporal continuity has little relevance.

Interrupts signifies either the occurrence of an internal operating system event such as the completion of a process or the status of the Real Time Systems clock, or, the readiness of a peripheral device such as a Teletype or a CRT to communicate data to or from the computer. The ability to respond to "external" events relative to the execution of the

current process allows processors to be shared by one or more processes if a means can be found to handle "simultaneous" events, i.e., those events occurring within one basic system cycle -- normally the current instruction cycle. The usual means of handling simultaneous "interrupt requests" is by embedding the "concurrent" processes within a priority structure. The priority structure can be implemented in hardware and/or software.

The functioning of an interruptable computer program can be viewed as similar to that of the job of a secretary. The secretary has a scheme of priorities about her work, higher priority items being serviced on a more immediate schedule than lower priority items. Imagine the situation of a letter being typed when the phone rings. The activity in progress, the letter, is suspended while the immediate demand of the telephone is serviced. When the phone call has been disposed of the former, lower priority, typing activity is resumed. Some items have ultimate priority in this scheme, a fire alarm for example. No sane person bothers to answer the phone or even less to continue typing a letter when the building is burning down. Another important feature of the secretary's work environment is that nothing is of such priority that it must be done instantaneously. If the phone rings in the middle of a typed word or while the typist is taking a sip of coffee, the work is finished or the cup set down on the desk before picking up the phone. This illustrates an important feature of interruptable environments.

However the interruption is serviced, it must not cause a disruption of the former activity in such a way that it cannot be successfully resumed.

When a program is interrupted, the presumption is that the cause of the interruption is of some immediate priority, but not of such priority that the lower priority task being interrupted needs to be disrupted. One thing therefore needs to be understood at the outset about interrupts. The instruction in progress when the interrupt request arrives is always finished before the interrupt is honored. It does not require much imagination to see what would happen if an addition or jump could be interfered with before being completed. When the lower priority activity was resumed after servicing the interrupt the program would have no means of rectifying the damage done by the half-completed addition or jump.

The most important single thing that the programmer must remember about interrupted programs is that the status of the interrupted program must be preserved. The program which services the high priority interrupting activity will use the same registers and flags to accomplish its task as the interrupted program uses. These registers and flags must be restored to their condition at the time of the interrupt before returning control to the former activity, or the former activity will be disrupted. Failing to observe this caution is the most common single error in programming for interrupt

driven systems. The saving and restoring of the status of the interrupted program is of crucial importance.

A very common type of interrupt is that caused by some event external to the program which does not require that data be transmitted. The event itself constitutes the required information. In this category are such applications as traffic counters. The passage of a car through the sensor of an expresway ramp does not require the transmission of data for that car to be counted in the flow. In this case the interrupt itself constitutes notice to the system that a car has passed the sensor and that the counter is to be incremented. A similar situation is encountered in devices that register angular position through the counting of passing gear teeth. Exactly which tooth has passed is not of any interest, only the fact that a tooth has passed is of consequence.

Perhaps the most common of these event counting situations occurs with the computer option called a real time clock. The real-time clock is not a clock in the common sense of that word. It does not keep time at all, but simple generates a series of pulses at uniform intervals, these pulses being used to cause interruption of the operating program at these uniform intervals. The program which counts the interrupts can use this counting to keep a programmed time-of-day clock.

The elementary process that is crucial to an interrupt structure is the CALL/RETURN transfer-of-control process.

The subroutine is a sequence of code that is executed upon the invocation of its name and that returns control to the calling sequence upon completing its execution. An interrupt process can be thought of as an unexpected or surprise subroutine call. In a program, the invocation is accomplished by inserting a call instruction at a known position in the instruction sequence. During interrupt processes, the invocation will occur at unknown positions in the control sequence. Thus, provision must be made for saving the return address in a known location for later retrieval. Mathematics can be described as a "replacement" process in which the replacements are made under control of the mathematician. Interrupt systems are those in which the replacement of a given control sequence by another can be made upon request from any external system. The complete control sequence is composed of a set of elementary sequences, or control strings, that can be edited by real-world systems to adapt to local conditions.

Varieties of interrupt structures are designed with one goal in mind: to share one CPU efficiently between several "concurrent" processes. This can be accomplished via this procedure:

- save state of current process;
- identify device requesting service;
- transfer control of CPU to this device;
- upon completion of service, restore state; and
- transfer control of CPU back to interrupted process.

Although minor variations exist in implementation of these steps, they are always executed. This procedure is shown diagrammatically in figure 24.

B. PRIORITY INTERRUPTS

There are two basic implementation strategies for priority interrupts: Polled priority interrupts and Vectored priority interrupts.

1. POLLED PRIORITY INTERRUPTS

Polled priority interrupt methods trap (acknowledge and jump) all interrupts requests to a common location, and a routine that POLLS status bits determines the source of the interrupt request. If the interrupting devices can be arranged in a hierarchical order, then the highest priority device will be polled first, the next highest will be polled second, and so on. Thus, if two devices request service at once, the higher priority will be encountered first in the poll and it will receive service first. It should be noted that this method does not involve clearing the interrupt request.

2. VECTORED PRIORITY INTERRUPTS

An interrupt system in which the hardware supplies a separate address for each interrupting device is called a VECTORED interrupt structure as opposed to the POLLED structure in which all devices trap to the same address, and device identification and conflict resolution are accomplished in software.

INTERRUPT PROCESS SCHEME

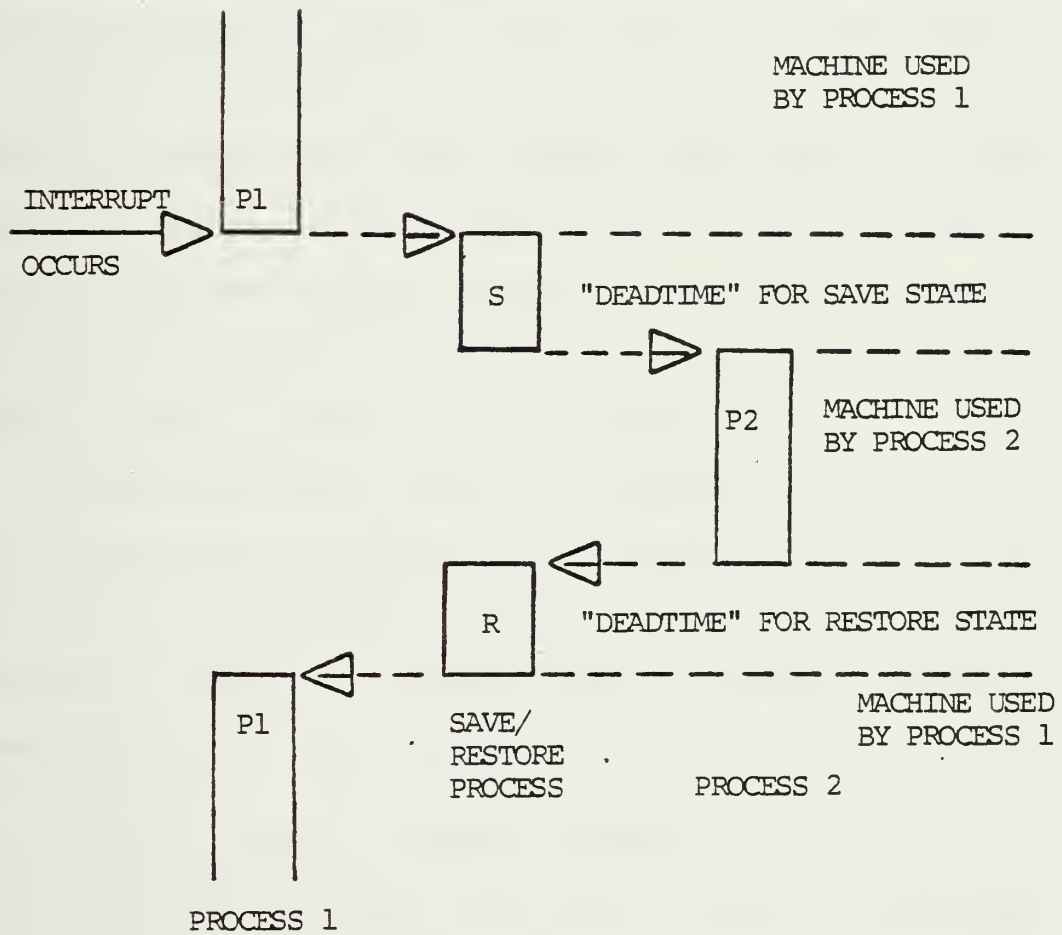


FIGURE 24

The use of hardware to encode these instructions for separate devices will speed up interrupt servicing by eliminating the need to POLL the devices. In addition, standard hardware is available for conflict resolution.

If two devices simultaneously request service, a priority encoder will pass only the higher priority request. This feature by which higher priority devices can interrupt lower priority devices, but lower priority devices cannot interrupt higher, is common to most interrupt structures.

Each device can have a unique address associated with its service routine, and the hardware just described automatically provides a 1-byte call instruction that causes transfer of control to this address. VECTORED interrupt systems provide the fastest possible interrupt servicing, because no time is wasted polling status bits.

Since the highest priority requests override all others there must be a means of individually removing each request as it is serviced, so that lower priority requests can be seen. The lower requests must, therefore, remain active until their time comes.

C. 8080 MICROPROCESSOR INTERRUPT METHOD

The 8080 microprocessor interrupt method is described below to illustrate a VECTORED PRIORITY INTERRUPT STRUCTURE implemented in a microcomputer.

An interrupt can occur when all of the following three conditions are met. If any of them is not met the interrupt cannot occur. They are:

1. The 8080's interrupt system has been enabled by the use of EI (Enable Interrupts) instruction. The term enable has specific application to the interrupt system as a whole and is not applicable to any specific device or peripheral. The interrupt system is enabled by the EI instruction and disabled by the DI (Disable Interrupts) instruction. In the disabled state no interrupts from any source can occur.

2. The specific device or peripheral interface has been conditioned by the program in such a way as to be able to generate interrupting pulses. This conditioning is known as arming. An interface which has been so conditioned is said to be armed. If the interface has not been conditioned so as to be able to generate interrupting pulses it is said to be disarmed.

3. The device or peripheral ready flag is set by the event which is to cause the interrupt.

Again, in the absence of any of the above conditions there is not and cannot be an interrupt. When the above conditions are all met, the following sequence of events takes place:

a. After finishing the execution of the instruction in progress at the time all of the conditions for interrupt were met, a special instruction is forced into the instruction register (I). This instruction is provided by the hardware itself and is not resident in computer memory. The program counter is not changed by this. It still points to the next program instruction to be executed, i.e., the

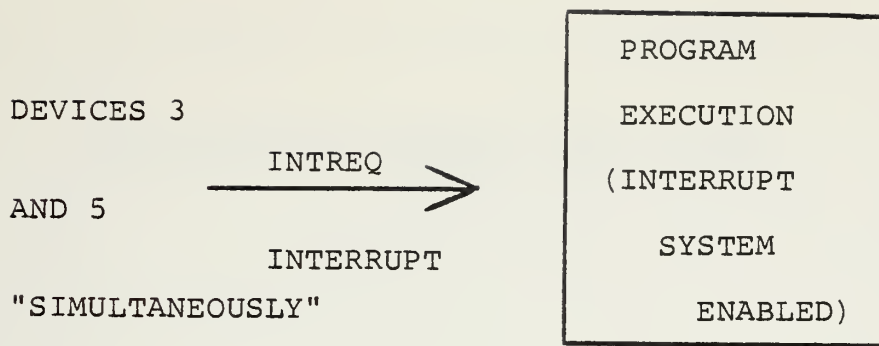
instruction following the one in progress at the time the interrupt conditions are met.

b. The special instruction forced into the instruction register is executed. This special instruction is a kind of CALL, but one in which the effective address is implicit rather than the normal CALL whose target address is explicitly given in the instruction. The special call, known as a restart, pushes the program counter onto the stack just as a normal CALL does. Instruction execution then starts at the address which was implicit in the interrupted instruction.

EXAMPLE: 8080 SOFTWARE INTERRUPT

The following was abstracted from "Microprocessor System Design," by C.E. Klingman [Reference 1]. In the example the Intel 8080 interrupt system is described in detail in order to present the hardware and software concepts associated with such systems. The concepts presented include POLLED versus VECTORED systems, priority determination and conflict resolution, priority thresholds, stacked state vectors, individually CLEARable device flags, and service routines. With minor variations, almost all interrupt systems utilize these concepts.

SCENARIO: During a process (i.e., service routine), I/O devices 3 and 5 interrupt almost simultaneously and then device 2 and then device 7 interrupt.



The lines from device 3 and device 5 clock the appropriate flip-flops, and the "REQUEST ACTIVITY" line becomes active. We assume that the comparator override is active and the binary code "5" from the encoder causes a high input to appear at the Interrupt Request flip-flop D-input. The next system clock pulse causes an interrupt request to be presented to the 8080 CPU. This same signal also latches the "5" into the priority latch and disables the gate following the comparator, thereby preventing further interrupts.

The CPU will finish the current execution and then, at the beginning of the following FETCH cycle, the INTAK status line will go high (and MEMRD will go low), causing the instruction to be fetched from the Interrupt Instruction Port instead of from memory. This port uses the output of the Priority Control Unit to compose a "TRAP 5" instruction that causes the (unincremented) program counter to be saved as a return address on top of the stack in the RAM and loads the program counter with 0...0101000. Thus, the next instruction is fetched from TRAP CELL 5 (location 40) in memory (see figure 25). This location contains a JUMP to FIVE-SERVICE, which is the routine as we now develop.

8080 EXAMPLE INTERRUPT FLOW DIAGRAM

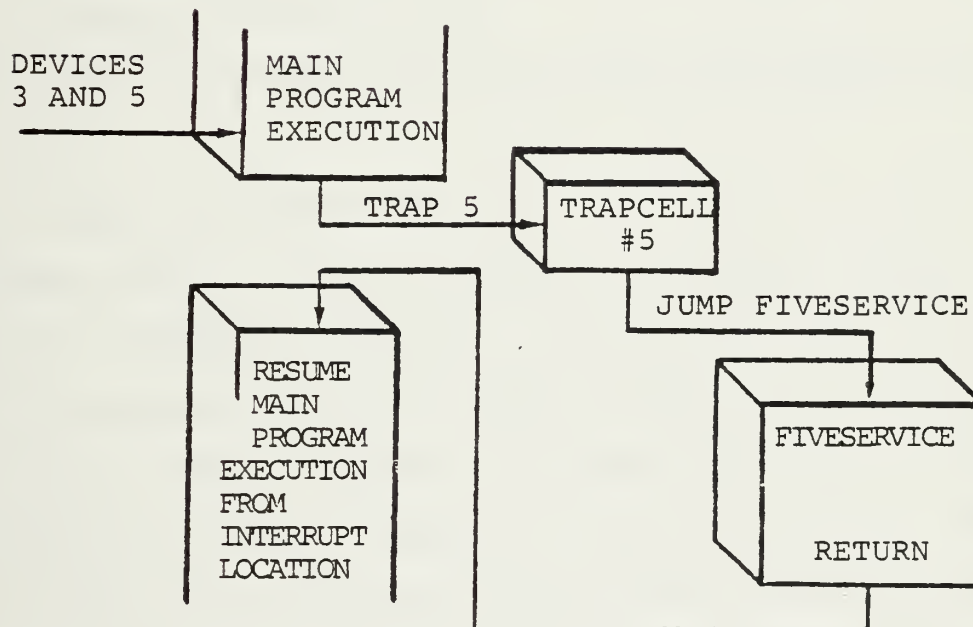


FIGURE 25

The service routine for device 5 performs the following tasks.

1. Push PSW, B,D,H (registers) onto the stack. The CPU state vector is saved for later return to the interrupted program execution.
2. Save the current status "IMAGE" by pushing it onto the stack.
3. Update the "IMAGE" in memroy.
4. Load the current status latch with 5, thereby enabling the 8214 Priority Control Unit, and clearing device 5 flag.
5. Enable the 8080 Interrupt system via execution of EI.
6. Begin servicing device 5.

Assume that at this point device 2 issues a request. Since the threshold is set at 5, this request will not generate an interrupt but will remain present as the device 2 flag. The 5 service routine continues executing. During its execution, device 7 issues an interrupt request. This code is above the threshold, and the 8214 PICU generates an interrupt to the 8080 CPU, latches #7 into AAA, and disables itself until enabled by the 8080. The hardware causes a TRAP 7 instruction to be composed and jammed into the CPU.

The location of the next instruction to be executed in FIVE-SERVICE is stored on the stack, and TRAP CELL 7 is entered. TRAP CELL 7 contains a JUMP SEVEN-SERVICE instruction, and control transfers to this routine. The 8080 state

vector (accumulator, flags, and register file) is pushed onto the stack, and the current "IMAGE" is also stored there. The Current Status Latch is updated, and the "IMAGE" is updated in memory. The device 7 flag is cleared when the current status is updated and the 8214 is re-enabled. The 8080 is re-enabled via an EI instruction, and device 7 is serviced. Upon completion of the device service, the 8080 is disabled via the DI instruction, and the "LAST STATUS" is POPped into the accumulator and restored to the current status latch, resetting the threshold to 5. Device 6 is assumed inactive; therefore, no INTREQ is generated by the 8214. The 8080 state vector is restored from the stack, the 8080 interrupt system is enabled, and a RETURN instruction is executed. (The system is enabled on the second instruction FETCH cycle following EI to prevent wild stack growth in a dense interrupt environment.) The return address stored on the stack is jammed into the program counter, and control returns to the interrupted FIVE-SERVICE routine.

Upon its completion, the CPU interrupt system is again disabled, the state vector restored from the stack, the IMAGE or "LAST STATUS" is POPped and loaded into the current status latch. The initial status was "ALL" (i.e., the comparator override bit was set); therefore, the device 3 flag that is still set causes the PICU to generate an interrupt request and a vector to TRAP CELL 3. The FIVE-SERVICE ends with an EI and a RETURN to the interrupted program. The

enabled 8080 interrupt system responds immediately to the pending request and enters TRAP CELL 3.

From this point, the THREE-SERVICE routine behaves exactly as the FIVE- and SEVEN-SERVICE routines. If no higher devices interrupt, THREE-SERVICE completes, returns to MAIN and finally device 2 flag gains entry to the TWO-SERVICE routine via TRAP CELL 2.

The procedure just described is summarized now for a general device in interrupt process.

1. Store JUMPT O N-SERVICE TO TRAP CELL N.
2. Jam TRAP N instruction into CPU via INTAK.

N-SERVICE:

1. Push CPU state vector onto stack.
2. Get "IMAGE" of status and save as "LAST STATUS" on the stack Update "IMAGE" in memory.
3. Load current status latch with N, clearing device N flag and re-enabling the 8214 Priority Interrupt Control Unit with threshold set to N.
4. Enable the 8080 interrupt system via execution of EI.
5. Service device N.
6. Disable 8080 interrupts (DI).
7. Reset threshold to "LAST STATUS" and update IMAGE in memory.
8. Re-store CPU state vector from stack.
9. Enable 8080 INT system (EI).
10. Return to interrupted location.
11. Honor highest pending interrupt, or proceed with interrupted routine if no interrupts are pending.

APPENDIX G

PLASMASCOPE TOUCH PANEL DESCRIPTION

The device electronics can be divided into three sections: the scanning system, consisting of the oscillator-counter; the light sources and the detectors; and the control logic.

The scanning system eliminates the optical collimation problem usually associated with a light-grid touch panel. The system only activates one light source/detector at a time. Time is the means of separating the light beams rather than a complex optical collimation system. This scanning is controlled by a free running oscillator driving a 4-bit counter. The output of the counter is used to sequentially select the light source/detector pairs, and to provide the address.

The light sources are infrared light emitting diodes (LED) chosen for their high output power, cost, and package design. Since these devices are diodes, a diode matrix drive scheme is used to reduce complexity. The output of the counter activates the appropriate transistors and causes two of the diodes to turn-on; one diode is in the x array, the other in the y array. In this way each diode pair (one x, one y) is sequentially pulsed and the display is scanned.

The detectors are silicon phototransistors similar to the LED in package design. The detectors are located across from the LED's in a plastic frame which fits around the display. Four detectors, spaced evenly along the side, share

a common amplifier. The output of the four amplifiers are time multiplexed so that the proper amplifier is actuated at the correct time. Only four amplifiers are needed because of the natural optical collimation associated with the plastic frame. For example, detectors #0, 4, 8, and 12 are activated when LED #0 is pulsed. Light from #0 LED is received by the #0 detector. The other detectors sharing the amplifier with #0 receive very little light from #0 LED.

The amplifier feeds the signal from the detector to a voltage comparator which can have one of two voltage thresholds. The purpose is to introduce scanning hysteresis into the system. Scanning hysteresis eliminates false inputs due to room light and partially broken beams. It works by setting up two conditions; one for initially detecting if the light beam is broken, and one for subsequently deciding when it is not broken. When initially scanning, the detect level is set low so that a beam to be detected broken must be completely absent. Upon the detection of both an x, y broken beam, the threshold voltage is raised to a higher level. Now a beam to be detected again must be larger than this higher threshold. In this way, marginal signals are ignored and only a beam, either absent or present is detected.

The basic operation of the total system is to sequentially activate pairs of source/detectors on both the x and y axes by means of the scanning logic. When a broken beam is detected, the address (or position) is stored in the appropriate storage register. When both an x and y beam are broken, the

information is sent to the computer. Scanning continues and with each scan, any new broken positions are compared with the old position stored in the registers. If the positions agree (i.e., the obstacle has not moved) scanning continues; if they disagree (the obstacle has been removed or shifted to a new location), the system resets. Touch inputs for the touch panel are limited to 10/sec by a short delay before reactivating. See figure 26 for Block Diagram.

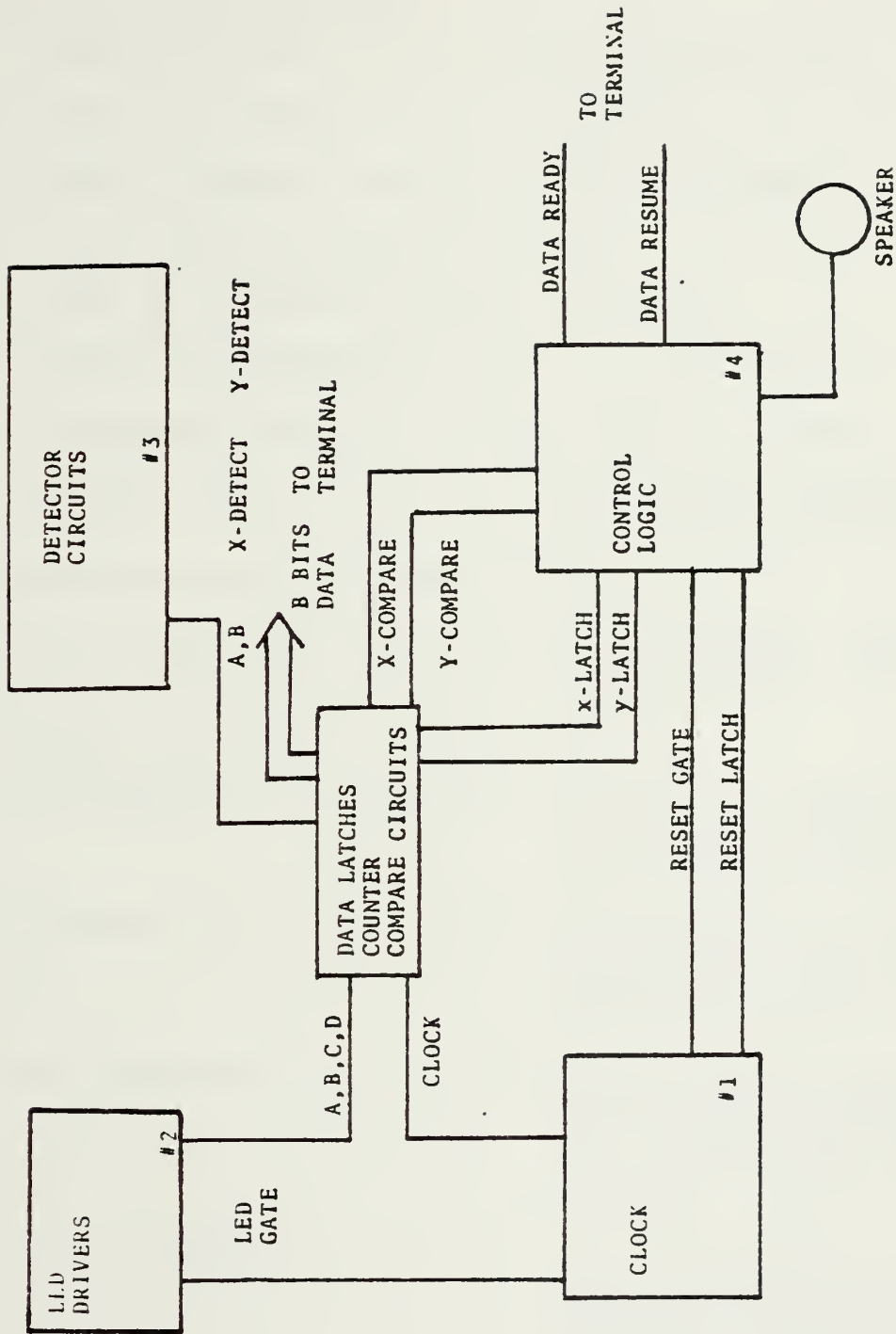


FIGURE 26

APPENDIX H

DATAMEDIA ELITE 2500 TERMINAL DESCRIPTION

A. ELITE 2500 SPECIFICATIONS

1.	SCREEN CAPACITY-----	1920 characters
2.	SCREEN TYPE/SIZE-----	P4 white, 12 inch
3.	SCREEN FILTER-----	Gray or Green
4.	CHARACTERS PER LINE-----	80
5.	LINES OF DISPLAY-----	Up to 24
6.	CHARACTER GENERATION-----	5 x 7 dot matrix
7.	CHARACTER SIZE-----	0.18"H x 0.09"W
8.	CHARACTER SET-----	Full ASCII - upper/lower case, 128 codes stored
9.	REFRESH RATES-----	50 or 60 Hz
10.	DATA RATES-----	50 to 9600bps, synchronous/ asynchronous
11.	MEMORY TYPE-----	MOS
12.	KEYBOARD-----	Electronic, typewriter layout with numeric cluster and cursor controls
13.	CURSOR-----	Addressable X-Y coordinates; non-destructive, non- blinking up, down, right, left, and home
14.	KEY CONTROLS-----	Erase screen, erase foreground data, erase to end of line, tab, print, Xmit, Xmit line, roll on, unlock, I/D, escape, return
15.	SPLIT SCREEN-----	Protected format, variable field tabbing horizontal and vertical. Blink field for computer derived or high light character
16.	OPERATING MODES-----	Full or Half Duplex, Roll or Tape Mode

17.	ALARM-----	Audible on alarm code or eight characters from end of line		
18.	INTERFACE-LINE -----	RS232C; lamp indicators for carrier detect, and clear to send (optional telegraphic interface)		
19.	INTERFACE-PRINTER-----	Independent printer output, speed preselectable		
20.	VIDEO OUTPUT-----	Provision to drive up to 16 external monitors		
21.	DIMENSIONS-----	14 $\frac{1}{2}$ "H x 20"D x 18"W		
22.	WEIGHT-----	50 pounds		
23.	POWER-----	100/125V	50/60Hz	150 Watts
		200/250V	50/60Hz	150 Watts

B. DESCRIPTION OF CONTROL FUNCTIONS

1. Home (STX) - Will return the cursor to the Home position (first character), first line), unless a protected field is located at home, then it will return cursor to the start of first variable field (end of first protected field).

2. Forward Cursor (FS) - A non-destruct code which will cause the cursor to advance one character position to the right. If there are no character positions to the right, the cursor will advance to the first character position of the next line. If position to the right is a protected field, the cursor will advance to the first character of the next variable field.

3. Back Cursor (BS) - A non-destruct code which will cause the cursor to move one character position to the left. The cursor will not move when it has reached the first character of the line or variable field.

4. Up Row Cursor (SUB) - A non-destruct code which will cause the cursor to move up one row. When the first row is reached, the cursor will remain there. If by moving the cursor up causes it to fall into a protected field, it will then advance to the first character of the next variable field.

5. Down Row Cursor (LF) - A non-destruct code which will move the cursor down one row. When the last row is reached it will move the cursor to the first row. If by moving the cursor down causes it to fall into a protected field, it will then advance to the first character of the next variable field. The (LF) will not function immediately after an automatic new line or a return (CR) code.

6. Return (CR) - A non-destruct code which will move the cursor to the start of the next line or variable field. The return will not function after an automatic new line.

7. Erase (US) - This code will erase all variable field data and position the cursor in the first character of the first variable field. If there is no protected field, it will position the cursor to the home position.

8. Master Clear (RS) - This code will clear the entire memory and home the cursor.

9. Erase to End of Line (ETB) - This code will erase all variable field data from cursor to the end of the line.

10. Start Address (FF) - This code will place the unit in the X-Y addressing mode. The next character will be

character address, and the following character the row address. Data may be typed from that point in the normal manner.

11. Xmit (DC1) - This code will cause the transmission of all data starting at the position of the cursor to the end of the page. All transmissions can be stopped by depressing the Break Key.

Transmission Identifiers

- a. Protected field will be identified by the transmission of (S1) ON protected field (CAN) OFF protected field. The field data will not be transmitted.
- b. Blink field will be identified by (S0) ON Blink field (CAN) OFF Blink field. Blink field characters are transmitted.
- c. Program options:
 - (1) At the end of each line a carriage return or (CR) (LF) can be sent.
 - (2) (DC1) (DC2) can cause an automatic home.
 - (3) (DC3) can cause an automatic carriage return.
 - (4) The transmission can be blocked by (STX) at the start and (ETX) at the end.
- d. Tape Mode Transmission - Transmits everything without adding any identifiers.

12. Xmit Line (DC3) - This code will cause data to be transmitted from the cursor to the end of the line, with the same transmission identifiers as (DC1).

13. Xmit Printer (DC2) - This code will cause all data to be transmitted to the printer starting at the cursor to the end of the page, adding a carriage return and line feed, and ten rub-out characters at the end of each line. This transmission is done, disregarding any form or Blink field that may exist.

14. Bell (BEL) - When this code is received the alarm will sound. The alarm will sound when the eighth character from the end of the line has been entered from the keyboard, or after a Data Transmission but not a Tape Mode Data Transmission.

15. Form On (SI) - This code will cause all characters sent after it to be in a protected field.

16. Blink On (S0) - This code will cause all characters sent after it to Blink.

17. Roll On (GS) - This code will cause the display to roll up when the cursor is on the last row and a (LF) or (CR) or 80th character is entered. It functions as an interactive mode, terminal to computer and display.

18. Form/Blink/Roll/I/D (CAN) - This code will cause the end to insert protected field, insert Blink, Roll Mode and insert delete.

19. Tab (HT) - When this code is received the cursor will move to the beginning of the next variable field. If no variable fields are found before the Home position, the cursor will stop tabbing at the Home position.

20. Insert Delete (DLE) - (optional) When this code is received, all data following will be entered at the cursor moving existing data over.

C. STANDARD FEATURES

1. PARITY CHECK AND GENERATION

Transmits even parity or mark parity by adding programming Jumper C to D on Transmitter J5. Receiver checks even parity, displaying question mark (?) character if parity errors is detected. The receive parity checker can be removed by removing Jumper AA to BB on receive board J4.

2. SPEED SELECTION

Defined data rates with five data rates selectable or with transmit and receive differentiating in SEL position.

3. FULL OR HALF DUPLEX

Identified by panel lights.

4. ROLL OR PAGE

Identified by panel lights.

5. PAGE TRANSMIT

Full page transmit starting at the cursor.

6. LINE TRANSMIT

Line by line transmit.

7. CURSOR CONTROL

Non-destructive, up, down, right, left, home.

8. AUDIBLE END OF LINE INDICATOR

Audible end of line bell sounds when the 72nd character of a line is entered; also sounds when the "Bell" character is received.

9. LINE CONDITIONS

Lamp indicator on front panel for (CD) carrier detect and (CTS) clear to send.

10. FORM FIELD CHARACTERS

Form field or protected field characters.

11. BLINK FIELD CHARACTER

Blink field character for highlight.

12. CARRIAGE RETURN

Generates automatic carriage return and line feed.

13. X-Y COORDINATES

X-Y coordinates cursor positioning.

14. 105 KEY-KEYBOARD

Not all used.

15. ENCODING OF ALL CONTROL FUNCTIONS

16. 127 AXCII CODES

Stored in tape mode.

D. OPTIONAL FEATURES

1. Current Loop Interface

2. Built-in 1200 Baud Modem

3. Separate Keyboard

4. Printer Xmit

5. Polled, up to 96 terminals can share a single communications line. (May be accommodated)

6. Character and Line Insert Delete

APPENDIX I

OPERATION OF THE "MDSPDP" INTERFACE PROGRAM

NOTE:

It is assumed that the operator is familiar with the Microcomputer Development System (MDS) operation and log-on procedures to the ISIS-II operating system, and log-on procedures to Unix. The operator must have a copy of the MDSPDP interface program in his floppy disk, and a copy of PDPSEND and PDPRECEIVE programs in his Unix directory. After MDS system initialization in the ISIS-II operating system, in order to execute the MDSPDP interface the operator must:

1. Issue the ISIS-II command

```
:FX:MDSPDP      :FX:FILENAME<CR>
```

This command causes the execution and subsequent transfer of system control to the MDSPDP program;

- :FX: is the designator of the drive in which the respective programs are located, X=0 for drive 0, X=1 for drive 1;
- (FILENAME) is the name of a particular file to be received or transmitted to or from the MDS. The name should be padded with blanks if it is not 11 characters long; and
- <CR> is carriage return.

It should be noted that once a file (FILENAME) is entered as an argument with an execution of MDSPDP ..., all transmissions

and receptions to or from the MDS will be to this file. This is especially important when exercising the "round-robin" feature of the interface between the two computer environments.

After entering the command, the CRT will display an echo-back of the :FX:FILENAME, and the status message:

SYSTEM IN NEUTRAL STATE:

and will proceed within 3 to 5 seconds to issue Unix log-in message:

NPS UNIX

NAME:

At this point the operator may enter his (her) user code and password (use lower case). Upon completion of the log-on procedure and prior of execution of PDPSSEND or PDPRECEIVE, the operator may utilize the Unix operating system from the MDS CRT terminal as a normal Unix time-sharing terminal.

2. Now, the operator is ready to transfer files:

a. PDPRECEIVE (transfer a file from floppy disk to PDP): to initiate transfer of a file, the operator must type the following command after the Unix prompt:
o/o.

o/o PDPRECEIVE <FILENAME> <CONTROL/T>

- <FILENAME> is the name of the file being transferred to the operator's Unix directory. The operator has the option of maintaining the original filename of his floppy disk directory file or change the name to a new one; in either case the file or the filename in the floppy disk would not be altered. During the transfer of data, the file will be echoed to the CRT. If the operator desires to abort the transmission he must depress:

<CONTROL / C> <REPEAT>

until the MDSPDP program print to the CRT the following message:

TRANSMIT STATE TERMINATED BY OPERATOR.

PARTIAL FILE CREATED AT PDP.

XXX BYTES TRANSMITTED FROM FLOPPY DISK TO PDP.

LEAVING TRANSMIT STATE: ENTER NEUTRAL STATE SYSTEM IN
NEUTRAL STATE

NEUTRAL STATE AND REBOOTING TO ISIS-II

- (prompt of ISIS-II)

xxx is the number of bytes transmitted so far. At this point the operator is back in the ISIS-II operating system, but the user is still logged-on to the PDP unix operating system. The operator must go to step one, initiate and complete a successful transmission, or go to step one, quit from Unix, and type the following command:

<CONTROL / C >

this will generate the following message,

NEUTRAL STATE AND REBOOTING TO ISIS-II

- (prompt of ISIS-II)

If the transmission is successfully completed, the operator will have in his Unix directory, the complete file with the name specified when he issued the command

o/o PDRECEIVE <FILENAME> <CONTROL / T> ,

After a successful transmission the following messages are printed to the CRT:

XXX BYTES TRANSMITTED FROM FLOPPY DISK TO PDP.

LEAVING THE TRANSMIT STATE: ENTER NEUTRAL STATE.

SYSTEM IN NEUTRAL STATE:

o/o (Unix prompt)

xxx is the number of bytes of the file transmitted.

At this time the operator may quit Unix, or operate

the MDS CRT as a Unix terminal (in the round-robin mode or as indicated in step one), editing the file,

then executing PDPSSEND (step 2b) to complete the

"round-robin" file transfer, and then quit Unix,

returning to the ISIS-II operating system, terminating MDSPDP program and data transmission.

b. PDPSSEND (transfer a file from the PDP to floppy disk).

After going through step one; to initiate transfer

of a file, the operator must type the following command after the Unix prompt: o/o

o/o PDPSEND <FILENAME> <CONTROL/ R>

- <FILENAME> is the name of the user's directory filename to be transferred, the operator has the choice of changing the name of the file to be transferred, this can be accomplished by calling the MDSPDP program (see step one), with the desired file name.

After this command is given the following message will appear in the CRT:

SYSTEM IN RECEIVE STATE:

and the transfer will begin from the file in Unix to the buffer in the MDS microcomputer. The file being transferred will not be echoed in the CRT until the complete file has been received in the buffer and the transfer from the buffer to the floppy disk is initiated, at this time, the echoed will start as well as the writing into the floppy disk. If the operator desires to abort the transmission of data, he must depress the following keys.

<CONTROL / C> <REPEAT>

(in this case no file will be written in the floppy disk). until the following messages appear in the CRT:

RECEIVE STATE TERMINATED BY OPERATOR.

NO FILE CREATED AT FLOPPY DISK.

NEUTRAL STATE AND REBOTTING TO ISIS-II.

- (ISIS-II prompt)

This will cause a termination of the MDSPDP program and a rebooting to the MDS ISIS-II operating system. At this point the operator must take the same action in case of transmission abort as explained in section 2.a

After successful transmission from the file at Unix into the MDS floppy-disk, and after receiving the echo of the file in the MDS-CRT the following messages will appear:

PDP PROMPTING : END OF RECEPTION.

PDP BUFFER/FILE WRITTEN TO DISK.

CHARACTER COUNT : XXX BYTES TRANSMITTED FROM PDP TO FLOPPY-DISK.

SYSTEM IN NEUTRAL STATE:

o/o (Unix Prompt)

At this time the operator may quit Unix, operate the MDS terminal as a Unix terminal (in the "round-robin" mode or as indicated in step-one), or execute PDPRECEIVE (step a) to complete the "round-robin" file transfer.

APPENDIX J
PROGRAM LISTINGS

A. STIS. MOD ----- 198

1. OUTPUTS\$STATUS\$LP
2. SEND\$CHAR\$LP
3. PCRLF
4. SEND\$STRING\$LP
5. INITIALIZE\$SCREEN
6. INITIALIZE\$PRINTER
7. CLOSE\$PRINTER
8. LIST\$MENU\$COMMANDS
9. LIST\$MENU\$COMMANDS\$LP
10. ORIG\$STRUCTURE\$WRITE
11. NEW\$STRUCTURE\$WRITE
12. ORIG\$PDP\$WRITE
13. MAKE\$ADDRESS
14. LOCATE\$STRUCTURE\$PDP
15. INT\$CHECK
16. INT\$3
17. INTE\$6
18. MAIN PROGRAM SOURCE CODE

B. SYSTEM. EXT ----- 200

1. OPEN
2. CLOSE
3. READ
4. WRITE

- 5. EXIT
- 6. CONSOL
- 7. DELETE
- 8. ERROR
- 9. RENAME

C. CRT.MOD -----202

- 1. SET\$TTY\$9600
- 2. SET\$TTY\$2400
- 3. OFF\$CRT\$KEYBOARD
- 4. ON\$CRT\$KEYBOARD
- 5. OUTPUT\$STATUS\$CRT
- 6. SEND\$CHAR\$CRT
- 7. CRLF
- 8. SEND\$STRONG\$CRT
- 9. PRINT\$TO\$CRT
- 10. INPUT\$STATUS\$PDP
- 11. PRINT\$CHAR\$PDP
- 12. INPUT\$STATUS\$CRT
- 13. READ\$CHAR\$CRT
- 14. SET\$LOW\$HOME
- 15. CLEAR\$LOW\$SCREEN
- 16. CLEAR\$NEXT\$LINES
- 17. GET\$BYTE
- 18. CLOCK\$TIMER

D. SCREEN.MOD ----- 272

- 1. WRITE\$BIG\$PICTURE
- 2. WRITE\$LITTLE\$PICTURE

3. CONVERT\$ADDRESS\$TO\$CHARTS
4. CONVERT\$BYTE\$TO\$CHARS
5. LOAD\$DATA\$BIG\$PICTURE
6. LOAD\$DATA\$LITTLE\$PICTURE
7. LOAD\$LINE\$ARRAY
8. BLANK
9. LOAD\$BLANKS
10. INITIALIZE\$LOAD\$PICTURE
11. SET\$STRUCTURE\$ZEROES

E. WRITE.MOD ----- 284

1. OUTPUT\$STATUS\$LP
2. SEND\$CHAR\$LP
3. LPCRLT
4. WRITE\$BIG\$PICTURE\$LP
5. WRITE\$LITTLE\$PICTURE\$LP

F. SCREEN.EXT ----- 213

1. CONVERT\$ADDRESS\$TO\$CHARS
2. CONVERT\$BYTES\$TO\$CHARS
3. WRITE\$BIG\$PICTURE\$LP
4. WRITE\$BIG\$PICTURE
5. WRITE\$LITTLE\$PICTURE\$LP
6. WRITE\$LITTLE\$PICTURE
7. LOAD\$DATA\$LITTLE\$PICTURE
8. INITIALIZE\$LITTLE\$PICTURE
9. SET\$STRUCTURE\$ZEROES

G.	GRAPH1MOD -----	313
1.	INITIALIZE\$GRAPHICS	
2.	SEND\$GRID\$PLASMA	
3.	SEND\$GRID\$PLASMA\$2	
4.	INITIALIZE\$STRUCTURE	
5.	DRAW\$SHIP\$DAST1	
6.	DRAW\$SHIP\$DASH\$LOOP	
7.	STRUCTURE\$BACK	
8.	BACKUPS	
9.	DRAW\$SHIP\$PROCEDURE	
H.	GRAPH2MOD -----	345
1.	DRAW\$SHIP\$DASH\$2	
2.	DRAW\$DASH\$LOOP\$-	
3.	DRAW\$SHIP\$2	
I.	SERVE.MOD -----	323
1.	SERVICE\$SIX	
2.	QUAD\$NUM	
J.	GRAPH1.EXT -----	217
1.	QUAD\$NUM	
2.	INITIALIZE\$GRAPHICS	
3.	INITIALIZE\$STRUCTURE	
4.	SEND\$GRID\$PLASMA	
5.	SEND\$GRID\$PLASMA\$2	
6.	DRAW\$SHIP\$DASH	
7.	DRAW\$SHIP\$DASH\$LOOP	
8.	STRUCTURE\$BACK	

- 9. BACKUPS
- 10. DRAW\$SHIPS
- 11. SERVICE\$SIX
- 12. DRAW\$SHIP\$DASH\$2
- 13. DRAW\$SHIP\$DASH\$LOOP\$2
- 14. DRAW\$SHIP\$2

K. PDP.MOD ----- 353

- 1. SYSTEM CALLS
- 2. OUTPUT\$STATUS\$CRT
- 3. SET\$TTY\$2400
- 4. OUTPUT\$STATUS\$PDP
- 5. INPUT\$STATUS\$CRT
- 6. INPUT\$STATUS\$PDP
- 7. SEND\$CHAR\$CRT
- 8. SEND\$CHAR\$PDP
- 9. CRLF
- 10. SEND\$STRING\$CRT
- 11. PRINT\$TO\$CRT
- 12. READ\$CHAR\$CRT
- 13. READ\$CHAR\$PDP
- 14. SET\$CHAR\$CRT\$BUF
- 15. GET\$CHAR\$PDP\$BUF
- 16. PUT\$CHAR\$CRT\$BUF
- 17. PUT\$CHAR\$PDP\$BUF
- 18. CRT\$BUF\$FULL
- 19. PDP\$BUF\$FULL
- 20. PRINT\$HEX\$NUMBER

- 21. FORMAT\$HEX
- 22. PRINT\$CHAR\$COUNT
- 23. INT\$CHAR\$COUNT
- 24. COUNT\$CHAR
- 25. INTE\$NEUTRAL\$STATE
- 26. INT\$RECEIVE\$STATE
- 27. INT\$TRANSMIT\$STATE
- 28. BREAK\$STATE
- 29. END\$R
- 30. END\$T
- 31. WRITE\$RECORD\$TO\$DISK
- 32. WRITE\$PDP\$BUFC
- 33. REBOOT
- 34. MAIN BODY OF PDP.MOD

L. PLASMA.MOD ----- 395

M. PLAPUB.ONE ----- 406

- 1. WRITE\$CONTACT\$ID
- 2. DRAW\$GRID
- 3. DRAW\$FRIEND\$SYMBOL
- 4. DRAW\$FRIEND\$DASH
- 5. ERASE\$FRIEND\$DASH
- 6. ERASE\$FRIEND\$SYMBOL
- 7. DRAW\$HOSTILE\$SYMBOL
- 8. DRAW\$HOSTILE\$DASH
- 9. ERASE\$HOSTILE\$SYMBOL
- 10. ERASE\$HOSTILE\$DASH

N. UNKNOWN.SR1 ----- 422

1. DRAW\$UNKNOWN\$SYMBOL
2. ERASE\$UNKNOWN\$SYMBOL
3. DRAW\$UNKNOWN\$DASH
4. ERASE\$UNKNOWN\$DASH

O. PLAPUB.TWO ----- 426

1. WRITE\$CONTACT\$ID\$2
2. DRAW\$GRID\$TWO
3. DRAW\$FRIEND\$SYMBOL\$2
4. DRAW\$FRIEND\$DASH\$2
5. ERASE\$FRIEND\$DASH\$2
6. ERASE\$FRIEND\$SYMBOL\$2
7. DRAW\$HOSTILE\$SYMBOL\$2
8. DRAW\$HOSTILE\$DASH\$2
9. ERASE\$HOSTILE\$SYMBOL\$2
10. ERASE\$HOSTILE\$DASH\$2

P. UNKNOWN.SR2 ----- 442

1. DRAW\$UNKNOWN\$SYMBOL\$2
2. ERASE\$UNKNOWN\$SYMBOL\$2
3. DRAW\$UNKNOWN\$DASH\$2
4. ERASE\$UNKNOWN\$DASH\$2

Q. PLAEXT.ONE ----- 395

1. SET\$STATUS\$PLASMA
2. PLASMA\$WRITE
3. CLEAR\$PLASMA
4. PLASMA\$WRITE\$VECTOR

5. PLASMA\$PRINT\$STRING
6. INITIALIZE\$PLASMA
7. SET\$VECTOR
8. START\$VECTOR\$SOLID
9. STOP\$VECTOR\$SOLID
10. START\$VECTOR\$DASH
11. STOP\$VECTOR\$DASH
12. GRAPHIC\$DESIGN
13. START\$ERASE\$VECTOR
14. STOP\$ERASE\$VECTOR
15. START\$ERASE\$DASH
16. STOP\$ERASE\$DASH

R. PLAEXT.TWO ----- 401

1. SET\$STATUS\$PLASMA\$ 2
2. PLASMA\$WRITE \$2
3. CLEAR\$PLASMA\$2
4. PLASMA\$WRITE\$VECTOR\$2
5. PLASMA\$PRINT\$STRING\$2
6. INITIALIZE\$PLASMA\$2
7. SET\$VECTOR\$2
8. START\$VECTOR\$SOLID\$2
9. STOP\$VECTOR\$SOLID\$2
10. START\$VECTOR\$DASH\$2
11. STOP\$VECTOR\$DASH\$2
12. GRAPHIC\$DESIG\$2
13. START\$ERASE\$VECTOR\$2
14. STOP\$ERASE\$VECTOR\$2

15. START\$ERASE\$DASH\$2

16. STOP\$ERASE\$DASH\$2

S. PLAPUB.EXT ----- 314

T. PSPRIM.SRC ----- 447

1. SETS\$STATUS\$PLASMA

2. PLASMA\$WRITE

3. CLEAR\$PLASMA

4. PLASMA\$WRITE\$VECTOR

5. PLASMA\$PRINT\$STING

6. INITIALIZE\$PLASMA

7. SET\$VECTOR

8. SET\$VECTOR\$ONE

9. START\$VECTOR\$SOLID

10. STOP\$VECTOR\$SOLID

11. START\$VECTOR\$DASH .

12. STOP\$VECTOR\$DASH

13. GRAPHIC\$DESIG

14. START\$ERASE\$VECTOR

15. STOP\$ERASE\$VECTOR

16. START\$ERASE\$DASH

17. STOP\$ERASE\$DASH

U. PSPR2.SRC ----- 469

1. SET\$STATUS\$PLASMA\$2

2. PLASMA\$WRITE\$2

3. CLEAR\$PLASMA\$2

4. PLASMA\$WRITE\$VECTOR\$2

5. PLASMA\$PRINT\$STRING\$2
6. INITIALIZE\$PLASMA\$2
7. SET\$VECTOR\$2
8. SET\$VECTOR\$ONE\$2
9. START\$VECTOR\$SOLID\$2
10. STOP\$VECTOR\$SOLID\$2
11. START\$VECTOR\$DASH\$2
12. STOP\$VECTOR\$DASH\$2
13. GRAPH1\$DESIGN\$2
14. START\$ERASE\$VECTOR\$2
15. STOP\$ERASE\$VECTOR\$2
16. START\$ERASE\$DASH\$2
17. STOP\$ERASE\$DASH\$2

V. STIS.C ----- 489

1. MAIN
2. INISTRU
3. INTO
4. SENDDATA
5. EOT
6. COMPUTE
7. LDLAT
8. LDLONG
9. LDCOUR
10. QUADNUM
11. LDRANG
12. LDBEAR
13. LDCPATD

W. MDSPDP.SRC ----- 516

1. HANDLER
2. PDPRECEIVE.C
3. PDPSEND.C

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SYSTEMMODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:STIS.MOD NOOBJECT PAGEWIDTH(82) PAGE
LENGTH(24) DAT
-E(MAY 1979) TITLE('MODULE STIS.MOD')

/* THIS MODULE CONTAINS PROCEDURES TO DRAW SYMBOLS TO
PLASMA DEVICES AND INTERACT WITH PLASMA TOUCH-PAN

THE

EL. */

1	SYSTEM\$MODULE:
	DO;
2	DECLARE LIT LITERALLY 'LITERALLY';
3	DECLARE DCL LIT 'DECLARE';
4	DCL TRUE LIT '0FFH';
5	DCL FALSE LIT '000H';
6	DCL LOC\$18H ADDRESS AT (18H);
7	DCL LOC\$19H ADDRESS AT (19H);
8	DCL LOC\$30H ADDRESS AT (30H);
9	DCL LOC\$31H ADDRESS AT (31H);
10	DCL STRUCTURE\$COUNT BYTE;


```

11      DCL (TEST$CHAR,IDA,IDB) BYTE;
12      DCL XY ADDRESS;
13      DCL Z BYTE;
14      DCL OK BYTE;
15      DCL PLASMA$SWITCH BYTE;
16      DCL PRINT$TEMP BYTE;
17      DCL COMMAND$MENU BYTE;
18      DCL SCREEN$PRINT BYTE;
19      DCL XY1 BYTE;
20      DCL XY2 BYTE;
21      DCL RECEIVE$PRINT BYTE PUBLIC;
22      DCL STR$NM (*) BYTE INITIAL (':F1:NEW.STR
23      DCL STR$OR$NM (*) BYTE INITIAL (':F1:ORIG.STR
);
);
24      DCL PDP$NM (*) BYTE INITIAL (':F1:ORIG.PDP
25      DCL (OPEN$STATUS,
        CLOSE$STATUS,
        CONSOL$STATUS,
        APT$IN,
        WRITE$STATUS) ADDRESS;

```



```

$INCLUDE(:F1:SYS.EXT)
/*ISIS-II SYSTEM CALLS: *****/

=
=
=
26 1 OPEN:
    PROCEDURE (AFT,FILE,ACCESS,MODE,STATUS ) EXTER
    NAL;
27 2 DECLARE (AFT,FILE,ACCESS,MODE,STATUS )AD
    DRESS;
28 2 END OPEN;

=
=
=
29 1 CLOSE:
    PROCEDURE(AFT,STATUS) EXTERNAL;
30 2 DCL (AFT,STATUS) ADDRESS;
31 2 END CLOSE;

=
=
=
32 1 READ:
    PROCEDURE(AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTE
    RNAL;
33 2 DCL(AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
34 2 END READ;

```



```

=
=
35 1        WRITE:
36 2            PROCEDURE(AFT,BUFFER,COUNT,STATUS) EXTERNAL;
37 2                DCL(AFT,BUFFER,COUNT,STATUS) ADDRESS;
     2        END WRITE;
=
=
38 1        EXIT:
39 2            PROCEDURE EXTERNAL;
40 2                DCL STATUS ADDRESS;
     2        END EXIT;
=
=
41 1        CONSOL:
42 2            PROCEDURE(INFILE,OUTFILE,STATUS) EXTERNAL;
43 2                DCL (INFILE,OUTFILE,STATUS) ADDRESS;
     2        END CONSOL;
=

```



```

=
=
44 1      DELETE:
45 2      PROCEDURE(FILE,STATUS) EXTERNAL;
46 2      DCL(FILE,STATUS) ADDRESS;
      END DELETE;
=
=
47 1      ERROR:
48 2      PROCEDURE (ERRNUM) EXTERNAL;
49 2      DCL (ERRNUM) ADDRESS;
      END ERROR;
=
=
50 1      RENAME:
51 2      PROCEDURE(OLDFILE,NEWFILE,STATUS) EXTERNAL;
52 2      DCL(OLDFILE,NEWFILE,STATUS) ADDRESS;
      END RENAME;
=
$INCLUDE(:F1:CRT.DEC)
=

```


/* THIS MODULE CONTAINS DECLARATIONS TO INTERACT WITH

THE MDS CRT . */

53 1 SET\$TTY\$9600: PROCEDURE;

54 2 /* SET TTY BAUD RATE TO 9600 */
55 2 OUTPUT(245)=40H;
56 2 OUTPUT(245)=4EH;
57 2 OUTPUT(245)=37H;
END;

58 1 SET\$TTY\$2400: PROCEDURE;

59 2 /* SET TTY BAUD RATE TO 2400 */
60 2 OUTPUT(245)=40H;
OUTPUT(245)=4FH;


```

61      2      OUTPUT(245)=37H;
62      2      END;

63      1      OFF$CRT$KEYBOARD: PROCEDURE;

                /* SET CRT USART RECEIVER TO DISABLE */
64      2      OUTPUT(247) = 40H;
65      2      OUTPUT(247) = 4FH;
66      2      OUTPUT(247) = 33H;
67      2      END; /* OFF$CRT$KEYBOARD */

68      1      ON$CRT$KEYBOARD: PROCEDURE;

                /* SET CRT USART RECEIVER TO ENABLE */
69      2      OUTPUT(247) = 40H;
70      2      OUTPUT(247) = 4FH;
71      2      OUTPUT(247) = 37H;
72      2      END; /* ON$CRT$KEYBOARD */

```



```

82      1      =      /* CRLF: */
      CRLF: PROCEDURE ;
      =
      =
      /* SEND CARRIAGE RETURN AND LINE FEED TO THE CRT *
/
83      2      =      CALL SEND$CHAR$CRT(0DH);
84      2      =      CALL SEND$CHAR$CRT(0AH);
85      2      =      END;
      =
      =
      =
86      1      =      SEND$STRING$CRT: PROCEDURE (STRING$ADDRESS);
      =
      =
      /* SEND MESSAGE AT STRING$ADDRESS UNTIL '$' IS
      DETECTED */
87      2      =      DCL STRING$ADDRESS ADDRESS;
88      2      =      DCL TEMPCHAR BASED STRING$ADDRESS BYTE;
89      2      =      DO WHILE TEMPCHAR <> '$';
90      3      =      CALL SEND$CHAR$CRT(TEMPCHAR);
91      3      =      STRING$ADDRESS=STRING$ADDRESS + 1;
92      3      =      END;
93      2      =      END;
```


207


```

104      =      /* READ A CHARACTER FROM THE PDP */
105      2      DCL CHAR BYTE;
106      2      CHAR= INPUT (244) AND 07FH;
107      2      /* CONVERT LOWER TO UPPER CASE */
108      2      IF (CHAR >=61H) AND (CHAR<=7AH) THEN
109      2      RETURN CHAR AND 0DFH;
          2      RETURN CHAR;
          2      END;
          =
          =
          =
110      1      INPUT$STATUS$CRT: PROCEDURE BYTE;
          =
          =      /* TRUE IF DATA INPUT LINE FROM CRT READY */
111      2      RETURN ROR(INPUT(247),1);
112      2      END;
          =
          =
          =
113      1      READ$CHAR$CRT: PROCEDURE BYTE;
          =
          =

```



```

/* THIS PROCEDURE LOCATES CURSOR AT COLUMN IN ROW 19.

```



```

130 2 = CALL SEND$CHAR$CRT(017H); /* CLEAR LOW HOME */
131 2 = END;
    =
    =
    =
132 1 = CLEAR$LOW$SCREEN: PROCEDURE;
    =
    = /* THIS PROCEDURE WILL CLEAR ROWS 19 THRU 24 */
133 2 DCL I BYTE;
134 2 DCL ETEOL LIT '17H';
135 2 DCL LF LIT '0AH';
136 2 CALL SET$LOW$HOME;
137 2 CALL SEND$CHAR$CRT(ETEOL);
138 2 DO I = 1 TO 4;
139 3 CALL SEND$CHAR$CRT(LF);
140 3 CALL SEND$CHAR$CRT(ETEOL);
141 3 END; /* DO I */
142 2 CALL SET$LOW$HOME;
143 2 END;
    =
    =
    =

```



```

144      1      =      CLEAR$NEXT$LINES: PROCEDURE(X);
      =
      =
      =      /* THIS PROCEDURE CLEARS X NUMBER OF NEXT LINES */
145      2      =      DCL (I,X) BYTE;
146      2      =      DCL ETEOL LIT '17H';
147      2      =      DCL LF LIT '0AH';
148      2      =      DO I = 1 TO X;
149      3      =          CALL SEND$CHAR$CRT(ETEOL);
150      3      =          CALL SEND$CHAR$CRT(LF);
151      3      =      END; /* DO I */
152      2      =      END;
      =
      =
      =
153      1      =      GET$BYTE: PROCEDURE(DIGITS) BYTE ;
      =
      =      /* THIS PROCEDURE IS USED TO OBTAIN A DECIMAL NUMBER B
      =      255 FROM THE CRT KEYBOARD. */
      =
154      2      =      DCL RUB LIT '07FH';
155      2      =      DCL BS LIT '08H';

```



```

156 2 = DCL BEL LIT '07H';
157 2 = DCL (NUMBER,DIGITS,CHAR,COUNT) BYTE;
158 2 = NUMBER,COUNT = 0;
159 2 = DO WHILE DIGITS > 0;
160 3 = CHAR = READ$CHAR$CRT;
161 3 = DO WHILE (((CHAR < '0') OR (CHAR > '9')) AND (CHAR
<> RUB)) OR
= ((CHAR = RUB) AND (COUNT = 0));
162 4 = CALL SEND$CHAR$CRT(BEL);
163 4 = CHAR=READ$CHAR$CRT;
164 4 = END; /* WHILE */
165 3 = IF CHAR <> RUB THEN
166 3 = DO;
167 4 = CALL SEND$CHAR$CRT(CHAR);
168 4 = NUMBER=NUMBER*10+(CHAR-30H);
169 4 = COUNT=COUNT+1;
170 4 = DIGITS=DIGITS-1;
171 4 = END; /*IF */
ELSE
DO;
172 3 = NUMBER=NUMBER/10;
173 4 = CALL SEND$CHAR$CRT(BS);
174 4 =

```



```

175 4 = COUNT=COUNT-1;
176 4 = DIGITS=DIGITS+1;
177 4 = END; /*ELSE */
178 3 = END; /* WHILE DIGITS */
179 2 = RETURN NUMBER;
180 2 = END GET$BYTE;
    =
    =
181 1 = CLOCK$TIMER: PROCEDURE (X) ;
    =
    = /* TIME DELAY : X IN SECONDS */
182 2 = DCL (X,Y) ADDRESS ;
183 2 = DO Y=1 TO (X*22);
184 3 = CALL TIME(250);
185 3 = END; /* DO Y */
186 2 = END; /* CLOCK$TIMER */
    = $INCLUDE(:F1:SCREEN.EXT)
    =
    = /* EXTERNAL DECLARATIONS FOR SCREEN.MOD */
187 1 = DCL CONVERTED$BYTE$NUMBER (5) BYTE EXTERNAL;
188 1 = DCL ERASE$COUNT ADDRESS EXTERNAL;

```



```

189 1 = CONVERT$ADDRESS$TO$CHARS: PROCEDURE (CHAR$ADDRESS) EXT
    ERNAL;
190 2 = DCL SCALE$FACTOR (5) ADDRESS DATA (10000,1000,100,10,1
    );
191 2 = DCL (JA,IA,PASS$COUNT) BYTE;
192 2 = DCL (CHAR,TEMP,CHAR$ADDRESS) ADDRESS;
193 2 = END;
    =
    =
    =
194 1 = CONVERT$BYTE$TO$CHARS: PROCEDURE (CHAR$BYTE) EXTERNAL;
195 2 = DCL SCALE$FACTOR (3) BYTE DATA (100,10,1);
196 2 = DCL (JB,IB,CHAR,CHAR$BYTE,PASS$COUNT,TEMP) BYTE;
197 2 = END;
    =
    =
    =
198 1 = WRITE$BIGPICTURE$LP: PROCEDURE EXTERNAL;
199 2 = DCL (I,J,K) ADDRESS;
200 2 = END;
    =
  
```



```

= =
= =
201 1 WRITE$BIGPICTURE: PROCEDURE EXTERNAL;
202 2 DCL (I,J,K) ADDRESS;
203 2 END;
= =
= =
204 1 WRITE$LITTLEPICTURE$LP: PROCEDURE EXTERNAL;
205 2 DCL (L,M,N) ADDRESS;
206 2 END;
= =
= =
207 1 WRITE$LITTLEPICTURE: PROCEDURE EXTERNAL;
208 2 DCL (L,M,N) ADDRESS;
209 2 END;
= =
= =
210 1 LOAD$DATA$LITTLEPICTURE: PROCEDURE (ILITTLE,JLITTLE) E
EXTERNAL;

```


211	2	=	DCL (I,J, ILITTLE,JLITTLE) BYTE;
212	2	=	DCL {N,K} BYTE;
213	2	=	END;
		=	
		=	
		=	
214	1	=	INITIALIZE\$LOAD\$PICTURE: PROCEDURE (C) EXTERNAL;
215	2	=	DCL (C,F) ADDRESS;
216	2	=	DCL (INDEX,SHIPINDEX,D,E) BYTE;
217	2	=	DCL T\$COUNT ADDRESS;
218	2	=	END;
		=	
		=	
		=	
		=	
219	1	=	SET\$STRUCTURE\$ZEROS: PROCEDURE (PTR,P\$COUNT) EXTERNAL
		=	;
220	2	=	DCL (PTR,P\$COUNT) ADDRESS;
221	2	=	DCL ZERO\$COUNTER ADDRESS;
222	2	=	DCL P BASED PTR BYTE;
223	2	=	END;
		=	
		=	

MAY 1

```

    =
    =
224 1  = DECLARE SHIP$PLOT (15) STRUCTURE (
        LAT(10) ADDRESS,
        LONG(10) ADDRESS,
        COURSE (10) ADDRESS,
        SPEED(10) BYTE,
        X$BOW(10) ADDRESS,
        Y$BOW(10) ADDRESS,
        QUADRANT(10) BYTE,
        RANGE(10) ADDRESS,
        BEARING(10) ADDRESS,
        COLLISION$FLAG(10) BYTE,
        CPA$TIME(10) ADDRESS,
        CPA$DISTANCE(10) ADDRESS,
        COUNT BYTE ) EXTERNAL ;
    =
    =
    =
225 1  = QUAD$NUM : PROCEDURE (X,Y) EXTERNAL;

```



```

226 2 = DCL (X,Y) ADDRESS;
227 2 = DCL QUADRANT$NUM BYTE ;
228 2 = END;
229 1 = INITIALIZE$GRAPHICS: PROCEDURE EXTERNAL;
/* THIS PROCEDURE INITIALIZES PLASMA DEVICES FOR GRAPH
ICS */
230 2 = END;
231 1 = INITIALIZE$STRUCTURE: PROCEDURE EXTERNAL;
/* THIS A PROCEDURE TO INITIALIZE STRUCTURE TO TEST MO
DULE STAND-ALO
NE. */
232 2 = DCL (A0,B0,C0,D0,E0,F0,G0,H0) ADDRESS;
233 2 = END; /* INITIALIZE$STRUCTURE */
234 1 = SEND$GRID$PLASMA: PROCEDURE EXTERNAL;

```


219


```

=
=
244 1 = STRUCTURE$BACK: PROCEDURE (A3) EXTERNAL;
=
=
0 NEXT
=
=
245 2 = ADJACENT BACKUP SHIP STRUCTURE SET. */
246 2 = DCL (A3,B,C,D,E) BYTE;
=
=
= END; /* STRUCTURE$BACK */
=
=
247 1 = BACKUPS: PROCEDURE (A4) EXTERNAL;
=
=
= /* THIS PROCEDURE TRANSFERS BACKUP STRUCTURES IN FIFO
QUEUE. */
248 2 = DCL (A4,B4,C4) BYTE;
249 2 = END; /* BACKUPS */
=
=
250 1 = DRAW$SHIP:PROCEDURE (TAIL) EXTERNAL;
=
=
= /*THIS PROCEDURE DRAW SHIPS IN THE PLASMA DISPLAY (DEV
ICE ONE) */

```



```

251 2 = DCL(A,TAI) BYTE;
252 2 = DCL (X1,Y1,X2,Y2,B,C ) ADDRESS;
253 2 = DCL (ZINDEX,ZSHIPINDEX) BYTE;
254 2 = END; /* DRAW$SHIP */
    =
    =
255 1 = SERVICE$SIX: PROCEDURE EXTERNAL;
    =
    = /* THIS PROCEDURE SERVICE THE INTERRUPTS CAUSED BY
    = THE PLASMA(DEVICE ONE) TOUCH-PANEL DEVICE. */
    =
256 2 = DCL (XY$DATAWORD,XY$DATAWORD$1,X$COORD,Y$COORD,INDEX,
    = SHIPINDEX,FOUND$SHIP,QUADRANT$NUM,OWNSHIP) BYTE;
257 2 = END; /* SERVICE SIX */
    =
    =
    =
258 1 = DRAW$SHIP$DASH$2: PROCEDURE (A1) EXTERNAL;
    =
    = /* THIS PROCEDURE DRAWS DASHED SHIPS IN THE PLASMA DIS
PLAY (DEVICE TWO). */
    =

```



```

259 DCL (A1,SHIPINDEX,INDEX) BYTE;
260 END;
    =
    =
    =
261 DRAW$SHIP$DASH$LOOP$2: PROCEDURE (A2) EXTERNAL;
    =
    =
    =
262 /* THIS PROCEDURE DRAWS BACKUP DASH SHIP POSITIONS */
263 DCL (A2,B2,C2) BYTE;
    =
    =
    =
264 END;
    =
    =
    =
265 DRAW$SHIP$2: PROCEDURE (TAIL) EXTERNAL;
    =
    =
    =
266 /* THIS PROCEDURE DRAW SHIPS IN THE PLASMA DISPLAY (DE
VICE TWO) */
267 DCL (A,TAIL) BYTE;
268 DCL (X1,Y1,X2,Y2,B,C) ADDRESS;
269 DCL (ZINDEX,ZSHIPINDEX) BYTE;
270 END;
    =
    =
    =

```



```

=
=
=
=
269 1 = DCL (MILI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,MONT
H,YEAR,SEC$TIME)
=
=
270 1 = BYTE PUBLIC;
271 1 = DCL TIME$STEP ADDRESS PUBLIC;
272 1 = DCL TIME$BUFFER(6) BYTE PUBLIC;
=
=
( TIME: : DATE: / /
);
=
273 1 = DCL M0 (73) BYTE EXTERNAL;
274 1 = DCL LOC$38 ADDRESS AT (038H);
275 1 = ICL LOC$39 ADDRESS AT (039H);
=
=
=
276 1 = INITIALIZE$M0$ARRAY: PROCEDURE ;

```



```

= =
277 2 /* THIS PROCEDURE INITIALIZES DATE/TIME FORMAT. */
278 2 DCL I BYTE ;
279 3 DO I=0 TO 72;
280 3 MC(I)=TIME$ARRAY(I);
281 2 END; /* DO I */
= =
= =
= =
= =
= =
282 1 CLOCK: PROCEDURE INTERRUPT 7;
= =
= =
M THE REAL TIM
E
283 2 CLOCK IN THE MDS SYSTEM, IS PRODUCED. */
284 2 DCL TEMP BYTE;
/* TO RESET THE MDS REAL TIME CLOCK. */
OUTPUT(0FFH) = 03H;

```



```

285 = MILI$SEC = MILI$SEC + 1;
286 = IF MILI$SEC = 128 THEN DO;
288 =   MILI$SEC = 0;
289 =   DUMMY$SEC = DUMMY$SEC + 1;
290 =   END; /* IF */
291 = IF DUMMY$SEC = 08H THEN DO;
293 =   MILI$SEC, DUMMY$SEC = 0;
294 =   SEC$TIME = 0FFH;
295 =   SECONDS = SECONDS + 1;
296 =   TIME$STEP = TIME$STEP + 1;
297 =   IF SECONDS = 60 THEN DO;
299 =     SECONDS = 00;
300 =     MINUTES = MINUTES + 1;
301 =     IF MINUTES = 60 THEN DO;
303 =       MINUTES = 00;
304 =       HOURS = HOURS + 1;
305 =       IF HOURS = 24 THEN DO;
307 =         HOURS = 00;
308 =         DAY = DAY + 1;
309 =         END; /* HOURS */
310 =       END; /* MINUTES */

```



```

311 4 = END; /* SECONDS */
312 3 = END; /* DUMMY$SEC */
313 2 = /* DISABLE INTERRUPTS */
314 2 = DISABLE;
315 2 = /* RESTORE CURRENT OPERATING LEVEL. */
316 2 = OUTPUT(0FFH) = 020H;
317 2 = /* SET THE MDS REAL TIME CLOCK. */
318 2 = TEMP = INPUT(0FFH);
319 2 = TEMP = INPUT(0FFH);
320 1 = OUTPUT(0FFH) = 00H;
321 2 = /* THE RETURN STATEMENT WILL ENABLE INTERRUPTS AUTOMAT
ICALLY. */
322 2 = RETURN;
323 2 = END; /* CLOCK */
324 1 =
325 1 =
326 1 =
327 1 =
328 1 = LOAD$TIME: PROCEDURE ;
329 1 = /* THIS PROCEDURE LOADS CURRENT TIME TO ARRAY 'M0' .
*/
330 1 = /* LOAD HOURS */
331 2 = CALL CONVERT$BYTE$TO$CHARS(HOURS);

```



```

322      M0(8)=CONVERTED$BYTE$NUMBER(1);
323      M0(9)=CONVERTED$BYTE$NUMBER(0);
      /* LOAD MINUTES */
324      CALL CONVERT$BYTE$TO$CHARS(MINUTES);
325      M0(11)=CONVERTED$BYTE$NUMBER(1);
326      M0(12)=CONVERTED$BYTE$NUMBER(0);
      /* LOAD SECONDS */
327      CALL CONVERT$BYTE$TO$CHARS(SECONDS);
328      M0(14)=CONVERTED$BYTE$NUMBER(1);
329      M0(15)=CONVERTED$BYTE$NUMBER(0);
      /* LOAD MONTH */
330      CALL CONVERT$BYTE$TO$CHARS(MONTH);
331      M0(24)=CONVERTED$BYTE$NUMBER(1);
332      M0(25)=CONVERTED$BYTE$NUMBER(0);
      /* LOAD DAY */
333      CALL CONVERT$BYTE$TO$CHARS(DAY);
334      M0(27)=CONVERTED$BYTE$NUMBER(1);
335      M0(28)=CONVERTED$BYTE$NUMBER(0);
      /* LOAD YEAR */
336      CALL CONVERT$BYTE$TO$CHARS(YEAR);
337      M0(30)=CONVERTED$BYTE$NUMBER(1);

```



```

338 2 = M0(31)=CONVERTED$BYTE$NUMBER(0);
339 2 = END; /* LOAD TIME */
=
=
=
=
=
340 1 = ERASE$LINE: PROCEDURE;
=
=
341 2 = /* ERASE PRESENT LINE */
342 2 = CALL SEND$CHAR$CRT(017H);
=
=
=
=
=
343 1 = CHECK$YES$NO: PROCEDURE BYTE;
=
=
/* THIS PROCEDURE USED TO CHECK FOR A VALID YES/NO INP
UT FROM THE CR
-
344 2 = DCI CHAR BYTE;
345 2 = CHAR=READ$CHAR$CRT;
346 2 = TO WHILE (CHAR <> 'Y') AND (CHAR <> 'N') /* UPPER CASE
*/

```



```

347 */
348 =
349 =
350 =
351 =
352 =
353 =
354 =
355 =
356 =
357 =
358 =
359 =
360 =

CALL SEND$CHAR$CRT(07H); /* BELL */
CHAR = READ$CHAR$CRT;
END; /* WHILE */
IF (CHAR='Y') OR (CHAR='y') THEN
DO;
CALL SEND$STRING$CRT(.( ' YES$' ));
RETURN 1;
END; /* IF */
ELSE
DO;
CALL SEND$STRING$CRT(.( 'NO $' ));
RETURN 0;
END; /* ELSE */
END; /* CHECK$YES$NO */

CHECK$INPUT: PROCEDURE BYTE ;

```



```

=      /* THIS PROCEDURE USED TO CHECK THE VALIDITY OF THE IN
PUT PRESENT AT
-      THAT
=      MOMENT IN THE SCREEN */
361 2   DCL CHAR BYTE;
362 2   CALL CRLF;
363 2   CALL SEND$STRING$CRT(.( 'IS INPUT DATA CORRECT? (Y/N) $
,));
364 2   CHAR=CHECK$YES$NO;
365 2   RETURN CHAR;
366 2   END; /* CHECK$INPUT */

=
=
=
367 1   BACK$SPACE$ERASE: PROCEDURE (X) ;
368 2   DCL (X,Y) BYTE ;
369 2   DO Y = 1 TO X;
370 3   CALL SEND$CHAR$CRT(08H);
371 3   END;
372 2   CALL ERASE$LINE;
373 2   END;
=
=

```



```

374 1  = INITIATE$TIME: PROCEDURE ;
    =
    =
    ME CLOCK. */
375 2  = DCL OK BYTE;
376 2  = DISABLE;
377 2  = OK = 0;
378 2  = DO WHILE OK = 0;
379 3  = YEAR, MONTH, DAY, HOURS, MINUTES, SECONDS = 0FFH;
380 3  = CALL SEND$STRING$CRT(.('INPUT DATE AND TIME. $'));
381 3  = CALL CRLF;
382 3  = DO WHILE YEAR >= 90;
383 4  = CALL SEND$STRING$CRT(.(' YEAR: $'));
384 4  = YEAR= GET$BYTE(2);
385 4  = IF YEAR >= 90 THEN CALL BACK$SPACE$ERASE(9);
387 4  = END; /* WHILE YEAR */
388 3  = DO WHILE MONTH >= 13;
389 4  = CALL SEND$STRING$CRT(.(' MONTH: $'));
390 4  = MONTH = GET$BYTE(2);
391 4  = IF MONTH >= 13 THEN CALL BACK$SPACE$ERASE(10);
393 4  = END; /* WHILE MONTH */
394 3  = DO WHILE DAY >= 32;

```



```
395 = CALL SEND$STRING$CRT(.( ' DAY: $' ));  
396 = DAY = GET$BYTE(2);  
397 = IF DAY >= 32 THEN CALL BACK$SPACE$ERASE(8);  
399 = END; /* WHILE DAY */  
400 = DO WHILE HOURS >= 24;  
401 = CALL SEND$STRING$CRT(.( ' HOURS: $' ));  
402 = HOURS = GET$BYTE(2);  
403 = IF HOURS >= 24 THEN CALL BACK$SPACE$ERASE(10);  
405 = END; /* WHILE HOURS */  
406 = DO WHILE MINUTES >= 60;  
407 = CALL SEND$STRING$CRT(.( ' MINUTES: $' ));  
408 = MINUTES = GET$BYTE(2);  
409 = IF MINUTES >= 60 THEN CALL BACK$SPACE$ERASE(12);  
411 = END; /* WHILE MINUTES */  
412 = DO WHILE SECONDS >= 60;  
413 = CALL SEND$STRING$CRT(.( ' SECONDS: $' ));  
414 = SECONDS = GET$BYTE(2);  
415 = IF SECONDS >= 60 THEN CALL BACK$SPACE$ERASE(12);  
417 = END; /* WHILE SECONDS */  
418 = OK = CHECK$INPUT;  
419 = CALL CLOCK$TIMER(2);
```


MAY 1

```

420      3      =      CALL CLEAR$LOW$SCREEN;
421      3      =      END; /* WHILE OK = 0 */
422      2      =      ENABLE;
      2      =
423      2      =      END; /* INITIATE$TIME */
      =
      =
      =      /* END DECLARATIONS FOR INTERRUPT DRIVEN CLOCK . */

424      1      OUTPUT$STATUS$LP: PROCEDURE BYTE;

      2      /* TRUE IF DATA OUTPUT LINE TO LP READY */
425      2      RETURN ROR(INPUT(245),2);
426      2      END;

427      1      SEND$CHAR$LP: PROCEDURE (CHAR);

```



```
428      /* SEND A CHARACTER TO THE LINE PRINTER */
429      DCL CHAR BYTE;
430      DO WHILE NOT OUTPUT$STATUS$LP;
431      END; /* DO WHILE */
432      OUTPUT(244)=CHAR;
433      END;

433      PCRLF: PROCEDURE;

434      /* SEND CRLF TO PRINTER */
435      CALL SEND$CHAR$LP(0DH);
436      CALL SEND$CHAR$LP(0AH);
437      END; /* PCRLF */

437      1      SEND$STRING$LP: PROCEDURE (STRING$ADDRESS);
```



```

D */
438      /* SEND MESSAGE AT STRING$ADDRESS UNTIL '$' IS DETECTE
439      DCL STRING$ADDRESS ADDRESS;
440      DCL TEMPCHAR BASED STRING$ADDRESS BYTE;
441      DO WHILE TEMPCHAR <> '$';
442          CALL SEND$CHAR$LP(TEMPCHAR);
443          STRING$ADDRESS= STRING$ADDRESS + 1;
444      END; /* TEMPCHAR */
      END; /* SEND$STRING$LP */

```

```

445 1 INITIALIZE$SCREEN: PROCEDURE;

```

```

/* THIS PROCEDURE PREPARES SCREEN FOR DISPLAY OF STATI

```

```

STICAL DATA .

```

```

- */
446 2 CALL SEND$CHAR$CRT(01EH); /* MASTER */
447 2 CALL SEND$CHAR$CRT(07H); /* RING BELL */
448 2 CALL SEND$CHAR$CRT(02H); /* HOME */
449 2 END;

```



```
450 1 INITIALIZE$PRINTER: PROCEDURE;
      A . */
451 2 CALL SET$TTY$9600;
452 2 OUTPUT(10)=0FFH;
453 2 CALL SEND$CHAR$LP(0CH); /* NEXT PAGE */
454 2 CALL SEND$CHAR$LP(0DH); /* CR */
455 2 CALL SEND$CHAR$LP(0AH); /* LF */
456 2 END;

457 1 CLOSE$PRINTER: PROCEDURE;
      PRINTER. */
458 2 CALL SEND$CHAR$LP(0CH);
459 2 CALL SEND$CHAR$LP(0CH);
460 2 OUTPUT(10)=000H;
461 2 CALL SET$TTY$2400;
462 2 END;
```

/* THIS PROCEDURE PREPARES PRINTER FOR PRINTING OF DAT

/* THIS PROCEDURE PAGES DATA THAT WAS JUST WRITTEN TO


```
463 1 LIST$MENU$COMMANDS: PROCEDURE;

464 2 /* THIS PROCEDURE DISPLAYS COMMAND OPTIONS */
465 2 PRINT$TEMP=SCREEN$PRINT;
466 2 SCREEN$PRINT=FALSE;
467 2 COMMAND$MENU=TRUE;
468 3 DO XY1=1 TO 2;
469 3 CALL CRLF;
470 2 END; /* XY1 */
471 2 CALL SEND$STRING$CRT(.
472 2 (,
473 2 CALL CRLF;
474 2 CALL CRLF;
475 2 CALL SEND$STRING$CRT(.
    ( 'B.....: DISPLAY GENERAL DATA STATISTICS
TO CRT SCREEN

                                COMMAND OPTIONS $' ));
                                ----- $' ));
```



```

- .$.');
476 2 CALL CRLF;
477 2 CALL SEND$STRING$CRT(
    ('G.....: DISPLAY GENERAL DATA STATISTICS
TO LINE PRINT
- ER.$'));
478 2 CALL CRLF;
479 2 CALL SEND$STRING$CRT(
    ('S.....: DISPLAY GENERAL DATA STATISTICS
PER DATA RECE
- PTION.$'));
480 2 CALL CRLF;
481 2 CALL SEND$STRING$CRT(
    ('L-F#,L-H#,L-U#...: DISPLAY CONTACT STATISTICS TO S
CREEN.$'));
482 2 CALL CRLF;
483 2 CALL SEND$STRING$CRT(
    ('P-F#,P-H#,P-U#...: DISPLAY CONTACT STATISTICS TO L
INE PRINTER.$
- ));
484 2 CALL CRLF;
485 2 CALL SEND$STRING$CRT(
    ('W.....: DISPLAY CURRENT TIME AND DATE T
O SCREEN.$'));
486 2 CALL CRLF;
487 2 CALL SEND$STRING$CRT(

```



```
      ('T.....: RESET DATE AND TIME.$');
488 2    CALL CRLF;
489 2    CALL SEND$STRING$CRT(.
      ('R.....: DATA RECEPTION VERIFICATION TO
SCREEN.$'));
490 2    CALL CRLF;
491 2    CALL SEND$STRING$CRT(.
      ('Z.....: SET STATUS OF PLASMA (1 AND 2)
AND LINE PRINT
- ER.$'));
492 2    CALL CRLF;
493 2    CALL SEND$STRING$CRT(.
      ('M.....: DISPLAY COMMAND OPTIONS TO SCRE
EN.$'));
494 2    CALL CRLF;
495 2    CALL SEND$STRING$CRT(.
      ('N.....: DISPLAY COMMAND OPTIONS TO LINE
PRINTER.$'));
496 2    CALL CRLF;
497 2    END; /* LIST$MENU$COMMANDS */

498 1    LIST$MENU$COMMANDS$LP: PROCEDURE;
```



```

/* THIS PROCEDURE DISPLAYS COMMAND OPTIONS TO LINE PRI
NTER */
499 2      CALL SEND$STRING$LP(
      (,
500 2      CALL PCRLF;
501 2      CALL SEND$STRING$LP(
      (,
502 2      CALL PCRLF;
503 2      CALL PCRLF;
504 2      CALL SEND$STRING$LP(
      ('B.....: DISPLAY GENERAL DATA STATISTICS
      TO CRT SCREEN.
      - $'));
505 2      CALL PCRLF;
506 2      CALL SEND$STRING$LP(
      ('G.....: DISPLAY GENERAL DATA STATISTICS
      TO LINE PRINT
      - R.$'));
507 2      CALL PCRLF;
508 2      CALL SEND$STRING$LP(
      ('S.....: DISPLAY GENERAL DATA STATISTICS
      PER DATA RECEP
      - TION.$'));
509 2      CALL PCRLF;

```

COMMAND OPTIONS \$'););

----- \$'););

DISPLAY GENERAL DATA STATISTICS

DISPLAY GENERAL DATA STATISTICS

DISPLAY GENERAL DATA STATISTICS


```
510 2      CALL SEND$STRING$LP(.  
      ('L-F#,L-H#,L-U#...: DISPLAY CONTACT STATISTICS TO SCR  
EEN.$')));  
511 2      CALL PCRLF;  
512 2      CALL SEND$STRING$LP(.  
      ('P-F#,P-H#,P-U#...: DISPLAY CONTACT STATISTICS TO LI  
NE PRINTER.$'))  
      );  
513 2      CALL PCRLF;  
514 2      CALL SEND$STRING$LP(.  
      ('W.....: DISPLAY CURRENT TIME AND DATE TO  
SCREEN.$')));  
515 2      CALL PCRLF;  
516 2      CALL SEND$STRING$LP(.  
      ('T.....: RESET DATE AND TIME.$')));  
517 2      CALL PCRLF;  
518 2      CALL SEND$STRING$LP(.  
      ('R.....: DATA RECEPTION VERIFICATION TO S  
CREEN.$')));  
519 2      CALL PCRLF;  
520 2      CALL SEND$STRING$LP(.  
      ('Z.....: SET STATUS OF PLASMA (1AND 2) AN  
D LINE PRINTER  
      -  
      .$')));  
521 2      CALL PCRLF;  
522 2      CALL SEND$STRING$LP(.  
      ('Z.....: SET STATUS OF PLASMA (1AND 2) AN  
D LINE PRINTER  
      -  
      .$')));
```



```

N.$'););
523 2 CALL PCRLF;
524 2 CALL SEND$STRING$LP(.
      ('N.....: DISPLAY COMMAND OPTIONS TO LINE
PRINTER.$'));
525 2 CALL PCRLF;
526 2 END; /* LIST$MENU$COMMANDS$LP */

527 1 ORIG$STRUCTURE$WRITE: PROCEDURE;
D AND /* THIS PROCEDURE WRITES THE ORIGINAL STRUCTURE CREATE
      STORED BY THE MDS MICRO . */

528 2 DCL (WI,ZK,ZJ) ADDRESS;
529 2 CALL OPEN(.AFT$IN,.STR$OR$NM,2,0,.OPEN$STATUS);
530 2 WI=128;
531 2 ZK=0;
532 2 DO ZJ = 1 TO 5;
533 3 IF ZJ = 5 THEN WI = 121;
535 3 CALL WRITE(AFT$IN,.SHIP$PLOT(0).LAT(0)+ZK,WI,.WRITE
$STATUS);

```



```

536      3      ZK = ZK + 128;
537      3      END; /* DO ZJ */
538      2      CALL CLOSE(AFT$IN,.CLOSE$STATUS);
539      2      END; /* ORIG$STRUCTURE$WRITE */

540      1      NEW$STRUCTURE$WRITE: PROCEDURE;

          /* THIS PROCEDURE WRITES THE MOST RECENTLY CREATED STR
          CRAFTED FROM THE PDP BUFFER. */
          DCL (NI,NK,NJ) ADDRESS;
          CALL OPEN(.AFT$IN,.STR$NM,2,0,.OPEN$STATUS);
          NI=128;
          NK=0;
          DO NJ = 1 TO 5;
              IF NJ = 5 THEN NI = 121;
              CALL WRITE(AFT$IN,.SHIP$PLOT(0).LAT(0)+NK,NI,.WRITE
$STATUS);
          NK=NK+128;
          END; /* DO NJ */
          CALL CLOSE(AFT$IN,.CLOSE$STATUS);

```



```

552      2      END; /* NEW$STRUCTURE$WRITE */

553      1      ORIG$PDP$WRITE: PROCEDURE;

INFORMATION
554      2      IN THE PDP BUFFER (MDS) FROM THE PDP-MINI. */
555      2      DCL (PI,PJ,PK) ADDRESS;
556      2      CALL OPEN(.AFT$IN,.PDP$NM,2,0,.OPEN$STATUS);
557      2      PI=128;
558      2      PK=0;
559      2      DO PJ = 1 TO 5;
560      3      IF PJ = 5 THEN PI = 121;
561      3      CALL WRITE(AFT$IN,.PDP$BUFFER(3)+PK,PI,.WRITE$STATU
S);
562      3      PK=PK+128;
563      3      END; /* DO PJ */
564      2      CALL CLOSE(AFT$IN,.CLOSE$STATUS);
565      2      END; /* ORIG$PDP$WRITE */

```


/* THIS ARRAY RECEIVES BYTE DATA IN THE MDS-MICRO FROM

THE PDP-MINI. */

566 1 DCL PDP\$BUFFER (2000) BYTE EXTERNAL;

567 1 PDP\$STRUCTURE: PROCEDURE EXTERNAL;

I */
568 2

END;

569 1 MAKE\$ADDRESS: PROCEDURE (X) ADDRESS;

/* THIS PROCEDURE CONVERTS TWO BYTE VALUE TO AN ADDRES

S VALUE */

570 2 DCL X ADDRESS;

571 2 DCL C ADDRESS;

572 2 C = PDP\$BUFFER(X+1);


```

573 2 C = (SHL(C,8)+PDP$BUFFER(X));
574 2 IF C >= 04000H THEN
575 2 DO;
576 3 C = C AND 03F7FH;
577 3 C = C OR 0080H;
578 3 END; /* IF C */
579 2 RETURN C;
580 2 END; /* MAKE$ADDRESS */

```

```

581 1 LOCATE$STRUCTURE$PDP: PROCEDURE;

```

246

```

/* THIS PROCEDURE LOCATES PDP BUFFER BYTE VALUES TO AP
PROPRIATE
SHIP PLOT STRUCTURE BYTE AND ADDRESS VALUES */
582 2 DCL (PDP$BUF$OFFSET,N,I,K,J,L) ADDRESS;
583 2 PDP$BUF$OFFSET=3;
584 2 DO I = 0 TO 2;
585 3 J=(I*211);
586 3 SHIP$PLOT(I).COUNT = PDP$BUFFER(J+210+PDP$BUF$OFFSE
T);
587 3 DO K = 0 TO 9;

```



```
588 4 N=K+J+PDP$BUF$OFFSET;  
589 4 L=(K*2)+J+PDP$BUF$OFFSET;  
590 4 SHIP$PLOT(I).LAT(K) = MAKE$ADDRESS(0+L);  
591 4 SHIP$PLOT(I).LONG(K) = MAKE$ADDRESS(20+L);  
592 4 SHIP$PLOT(I).COURSE(K) = MAKE$ADDRESS(40+L);  
593 4 SHIP$PLOT(I).SPEED(K) = PDP$BUFFER(60+N);  
594 4 SHIP$PLOT(I).X$BOW(K) = MAKE$ADDRESS(70+L);  
595 4 SHIP$PLOT(I).Y$BOW(K) = MAKE$ADDRESS(90+L);  
596 4 SHIP$PLOT(I).QUADRANT(K) = PDP$BUFFER(110+N);  
597 4 SHIP$PLOT(I).RANGE(K) = MAKE$ADDRESS(120+L);  
598 4 SHIP$PLOT(I).BEARING(K) = MAKE$ADDRESS(140+L);  
599 4 SHIP$PLOT(I).COLLISION$FLAG(K) = PDP$BUFFER(160+N)  
;  
600 4 SHIP$PLOT(I).CPA$TIME(K) = MAKE$ADDRESS(170+L);  
601 4 SHIP$PLOT(I).CPA$DISTANCE(K) = MAKE$ADDRESS(190+L)  
;  
602 4 END; /* DO K */  
603 3 END; /* DO I */  
  
604 2 END; /* LOCATE$STRUCTURE$PDP */  
  
605 1 INIT$CHECK: PROCEDURE;
```


606	2	DCL (D,E) BYTE;
607	2	DO E=0 TO 2;
608	3	SHIP\$PLOT(E).COUNT='L';
609	3	DO D=0 TO 9;
610	4	SHIP\$PLOT(E).LAT(D)='10';
611	4	SHIP\$PLOT(E).LONG(D)='32';
612	4	SHIP\$PLOT(E).COURSE(D)='54';
613	4	SHIP\$PLOT(E).SPEED(D)='6';
614	4	SHIP\$PLOT(E).X\$BOW(D)='87';
615	4	SHIP\$PLOT(E).Y\$BOW(D)='A9';
616	4	SHIP\$PLOT(E).QUADRANT(D)='B';
617	4	SHIP\$PLOT(E).RANGE(D)='DC';
618	4	SHIP\$PLOT(E).BEARING(D)='FE';
619	4	SHIP\$PLOT(E).COLLISION\$FLAG(D)='G';
620	4	SHIP\$PLOT(E).CPA\$TIME(D)='HI';
621	4	SHIP\$PLOT(E).CPA\$DISTANCE(D)='KJ';
622	4	END;
623	3	END;
624	2	END;


```
625 1 INT$3:PROCEDURE INTERRUPT 3;  
626 2 CALL OFF$CRT$KEYBOARD;  
627 2 Z =INPUT(244) AND 07FH;  
628 2 IF (Z = '2') THEN  
629 2 DO;  
630 3 IF STRUCTURE$COUNT > 0 THEN  
631 3 CALL BACKUPS (STRUCTURE$COUNT);  
632 3 CALL SET$LOW$HOME;  
633 3 CALL PDP$STRUCTURE;  
634 3 CALL LOCATE$STRUCTURE$PDP;  
635 3 IF (PLASMA$SWITCH=0) THEN  
636 3 DO;  
637 4 CALL INITIALIZE$PRINTER;  
638 4 CALL CONSOL(.(':CI:$'),.(':TO:$'),..CONSOL$STATUS)  
639 4 CALL LOAD$TIME;  
640 4 CALL WRITE$BIGPICTURE$LP;  
641 4 CALL CONSOL(.(':CI:$'),.(':VO:$'),..CONSOL$STATUS)  
642 4 CALL CLOSE$PRINTER;  
643 4 END; /* P=0 */  
644 3 IF ((PLASMA$SWITCH = 2) OR (PLASMA$SWITCH = 3)) THE
```

N


```
645 3 DO;
646 4 CALL SEND$GRID$PLASMA$2;
647 4 IF PLASMA$SWITCH = 3 THEN CALL DRAW$SHIP$2(0);
649 4 ELSE CALL DRAW$SHIP$2(1);
650 4 END; /* IF P=2 OR P=3 */
651 3 IF ((PLASMA$SWITCH=1) OR (PLASMA$SWITCH=3)) THEN
652 3 DO;
653 4 CALL SEND$GRID$PLASMA;
654 4 CALL DRAW$SHIP(1);
655 4 END; /* IF P=1 OR P=3 */
656 3 END;
      ELSE
657 2 DO;
658 3 DO WHILE (Z <> '%');
659 4 DO WHILE NOT INPUT$STATUS$PDP;
660 5 END;
661 4 Z = READ$CHAR$PDP;
662 4 END;
663 3 CALL PRINT$TO$CRT(.( ' ***** SKIPPED/MISSED INTERRUPT
! *****$ ));
664 3 END;
665 2 XY=2;
```


MAY 1

```

666 2 IF SCREEN$PRINT=TRUE THEN
667 2 DO;
668 3 CALL INITIALIZE$LOAD$PICTURE(XY);
669 3 CALL SEND$CHAR$CRT(Ø7H); /* BELL */
670 3 CALL SEND$CHAR$CRT(Ø2H); /* HOME */
671 3 CALL CLEAR$NEXT$LINES(19);
672 3 CALL SEND$CHAR$CRT(Ø2H); /* HOME */
673 3 CALL LOAD$TIME;
674 3 CALL WRITE$BIGPICTURE;
675 3 CALL CLEAR$LOW$SCREEN;
676 3 END;
677 2 ELSE IF RECEIVE$PRINT THEN
678 2 DO;
679 3 CALL CLOCK$TIMER(4);
680 3 CALL CLEAR$LOW$SCREEN;
681 3 END; /* RECEIVE$PRINT */
683 2 IF COMMAND$MENU=TRUE THEN
684 3 DO;
685 3 CALL CLEAR$LOW$SCREEN;
686 3 SCREEN$PRINT=PRINT$TEMP;
687 3 END; /* IF COMMAND$MENU=TRUE */

```



```

687 2 COMMAND$MENU=FALSE;
688 2 OUTPUT(243)=0FFH;
689 2 OUTPUT(253)=020H;
690 2 IF STRUCTURE$COUNT < 4 THEN
691 2     STRUCTURE$COUNT = STRUCTURE$COUNT + 1;
692 2     CALL SET$LOW$HOME;
693 2     CALL SEND$STRING$CRT(.( 'DATA UPDATED : READY . $\' ));
694 2     CALL ON$CRT$KEYBOARD;
695 2     DO XY1=1 TO 5;
696 3         CALL TIME(250);
697 3     END; /* XY1=1 TO 5 */
698 2     CALL ON$CRT$KEYBOARD;
699 2     ENABLE;
700 2     END; /* INTERRUPT THREE */

701 1 INT$6: PROCEDURE INTERRUPT 6;

     /* THIS PROCEDURE SERVICE THE INTERRUPTS CAUSED BY
     THE PLASMA(DEVICE ONE) TOUCH-PANEL DEVICE. */

```


MAY 1

```

702      2      CALL SET$LOW$HOME;
703      2      CALL SERVICE$SIX;
704      2      OUTPUT(243)=0FFH;
705      2      OUTPUT(253)=020H;
706      2      ENABLE;
707      2      END; /* INTERRUPT PROCEDURE INT$6 */

708      1      DISABLE;
709      1      CALL SET$TTY$2400;
710      1      MILI$SEC,DUMMY$SEC=00;
711      1      CALL INITIALIZE$GRAPHICS;
712      1      PLASMA$SWITCH = 3;
713      1      PRINT$TEMP=TRUE;
714      1      COMMAND$MENU = FALSE;
715      1      RECEIVE$PRINT=TRUE;
716      1      SCREEN$PRINT=TRUE;
717      1      ERASE$COUNT=0;
718      1      XY=3165;
719      1      CALL SET$STRUCTURE$ZER0ES(.SHIP$PLOT(0).LAT(0),XY);
720      1      STRUCTURE$COUNT=0;

```


MAY 1

```

721 1      OUTPUT(253)=12H;
722 1      OUTPUT(252)=0;
723 1      OUTPUT(252)=0B0H;
724 1      OUTPUT(243)=0FFH;
725 1      LOC$18H=0C3H;
726 1      LOC$19H=.INT$3;
727 1      LOC$30H=0C3H;
728 1      LOC$31H=.INT$6;
729 1      LOC$38 =0C3H;
730 1      LOC$39 =.CLOCK;
731 1      CALL MOVE(3,038H,008H);

732 1      CALL SEND$CHAR$CRT(018H);
/* OUT OF 'ROLL MODE' */
733 1      CALL SEND$CHAR$CRT(01EH); /* MASTER CLEAR */
734 1      CALL INITIALIZE$M0$ARRAY;
735 1      ENABLE;
736 1      OUTPUT(0FFH) = 00H;
737 1      CALL SET$LOW$HOME;
738 1      CALL INITIATE$TIME;
739 1      DO WHILE 1;
```



```

740 2 IF (ROR (INPUT(247),1)) THEN
741 2 DO;
742 3 TEST$CHAR=INPUT(246) AND 07FH;
743 3 IF TEST$CHAR = 'L' THEN
744 3 DO;
745 4 DISABLE;
746 4 CALL SET$LOW$HOME;
747 4 CALL SEND$STRING$CRT(.
      ('CONTACT STATISTICS TO SCREEN (Y/N) $'));
748 4 IF CHECK$YES$NO THEN
749 4 DO;
750 5 CALL CLOCK$TIMER(2);
751 5 CALL CLEAR$LOW$SCREEN;
752 5 OK=0;
753 5 DO WHILE OK=0;
754 6 CALL SET$LOW$HOME;
755 6 CALL SEND$STRING$CRT(.('CONTACT STATISTICS TO SC
REEN. $'));
756 6 CALL CRLF;
757 6 CALL SEND$STRING$CRT(.('ENTER CONTACT ID:F# OR H
# OR U# . $'));
758 6 DO WHILE NOT (ROR(INPUT(247),1));
```



```
759      7      END;  
760      6      IDA= INPUT(246) AND 07FH;  
761      6      DO WHILE  
762      7          ((IDA<>'h') AND (IDA<>'h') AND (IDA<>'u') AND  
763      7          (IDA<>'u') AND (IDA<>'f') AND (IDA<>'f'));  
ENTER CORRECT      CALL SEND$CHAR$CRT(07H);  
                     CALL SEND$STRING$CRT('(' INPUT DATA INCORRECT,  
                     DATA.$'));  
764      7      DO WHILE NOT(ROR(INPUT(247),1));  
765      8      END;  
766      7      IDA=INPUT(246) AND 07FH;  
767      7      CALL BACK$SPACE$ERASE(41);  
768      7      END; /* DO WHILE */  
769      6      CALL SEND$CHAR$CRT(IDA);  
770      6      IF IDA='f' OR IDA='f' THEN XY=0;  
772      6      IF IDA='h' OR IDA='h' THEN XY=1;  
774      6      IF IDA='u' OR IDA='u' THEN XY=2;  
776      6      DO WHILE NOT(ROR(INPUT(247),1));  
777      7      END;  
778      6      IDB = INPUT(246) AND 07FH;  
779      6      DO WHILE((IDB-'0') > (SHIP$PLOT(XY).COUNT-1) OR
```



```
780 7 (SHIP$PLOT(XY).COUNT = 0));
781 7 CALL SEND$CHAR$CRT(07H);
ENTER CORRECT CALL SEND$STRING$CRT(.( ' INPUT DATA INCORRECT,
-- DATA.$' ));
782 7 DO WHILE NOT(ROR(INPUT(247),1));
783 8 END;
784 7 IDB=INPUT(246) AND 07FH;
785 7 CALL BACK$SPACE$ERASE(41);
786 7 END; /* DO WHILE */
787 6 CALL SEND$CHAR$CRT(IDB);
788 6 OK=CHECK$INPUT;
789 6 CALL CLOCK$TIMER(2);
790 6 IF OK=0 THEN CALL CLEAR$LOW$SCREEN;
792 6 END; /* WHILE OK=0 */
793 5 CALL LOAD$DATA$LIT$PICTURE(IDA,IDB);
794 5 CALL INITIALIZE$SCREEN;
795 5 CALL LOAD$TIME;
796 5 CALL WRITE$LIT$PICTURE;
797 5 CALL CLEAR$NEXT$LINES(5);
798 5 CALL CLEAR$LOW$SCREEN;
799 5 END; /* IF CHECK$YES$NO */
```



```
800      ELSE
801      DO;
802      CALL CLOCK$TIMER(2);
803      CALL CLEAR$LOW$SCREEN;
804      END; /* ELSE */
805      ENABLE;
806      END; /* IF TEST$CHAR = 'L' */
807      ELSE
808      IF TEST$CHAR = 'P' THEN
809      DO;
810      DISABLE;
811      CALL SET$LOW$HOME;
812      CALL SEND$STRING$CRT(
813      ( 'CONTACT STATISTICS TO LINE PRINTER (Y/N) $' )
814      );
815      IF CHECK$YES$NO THEN
816      DO;
817      CALL CLOCK$TIMER(2);
818      CALL CLEAR$LOW$SCREEN;
819      OK=0;
820      DO WHILE OK=0;
821      CALL SET$LOW$HOME;
```



```
818 6      CALL SEND$STRING$CRT(.  
      ('CONTACT STATISTICS TO LINE PRINTER. $'  
);  
819 6      CALL CRLF;  
820 6      CALL SEND$STRING$CRT(.( 'ENTER CONTACT ID:F# 0  
R H# OR U# : $ - '));  
  
821 6      DO WHILE NOT (ROR(INPUT(247),1));  
822 7      END;  
823 6      IDA=INPUT(246) AND 07FH;  
824 6      DO WHILE  
      ((IDA<>'H') AND (IDA<>'h') AND (IDA<>'U') A  
      (IDA<>'u') AND (IDA<>'F') AND (IDA<>'f'));  
825 7      CALL SEND$CHAR$CRT(07H);  
826 7      CALL SEND$STRING$CRT(.( ' INPUT DATA INCORRECT  
,ENTER CORRECT  
      DATA.$' ));  
827 7      DO WHILE NOT(ROR(INPUT(247),1));  
828 8      END;  
829 7      IDA=INPUT(246) AND 07FH;  
830 7      CALL BACK$SPACE$ERASE(41);  
831 7      END; /* DO WHILE */  
832 6      CALL SEND$CHAR$CRT(IDA);  
833 6      IF IDA ='F' OR IDA ='f' THEN XY=0;
```



```
835 6 IF IDA = 'H' OR IDA = 'h' THEN XY=1;  
837 6 IF IDA = 'U' OR IDA = 'u' THEN XY=2;  
839 6 DO WHILE NOT(ROR(INPUT(247),1));  
840 7 END;  
841 6 IDB=INPUT(246) AND 07FH;  
842 6 DO WHILE ((IDB-'0') > (SHIP$PLOT(XY).COUNT-1) 0  
R  
843 7 (SHIP$PLOT(XY).COUNT = 0 ));  
844 7 CALL SEND$CHAR$CRT(07H);  
      CALL SEND$STRING$CRT('(' INPUT DATA INCORRECT  
      DATA.$'););  
845 7 DO WHILE NOT (ROR(INPUT(247),1));  
846 8 END;  
847 7 IDB=INPUT(246) AND 07FH;  
848 7 CALL BACK$SPACE$ERASE(41);  
849 7 END; /* DO WHILE */  
850 6 CALL SEND$CHAR$CRT(IDB);  
851 6 OK=CHECK$INPUT;  
852 6 CALL CLOCK$TIMER(2);  
853 6 IF OK=0 THEN CALL CLEAR$LOW$SCREEN;  
855 6 END; /* WHILE OK=0 */  
856 5 CALL LOAD$DATA$LIT$PICTURE(IDA,IDB);
```



```
857      CALL INITIALIZE$PRINTER;
858      CALL CONSOL(.(':CI:$'),.(':TO:$'),..CONSOL$STA
TUS);
859      CALL LOAD$TIME;
860      CALL WRITE$LIT$PICTURE$LP;
861      CALL CONSOL(.(':CI:$'),.(':VO:$'),..CONSOL$STA
TUS);
862      CALL CLOSE$PRINTER;
863      CALL CLEAR$LOW$SCREEN;
864      END; /* IF CHECK$YES$NO */
      ELSE
      DO;
865      CALL CLOCK$TIMER(2);
866      CALL CLEAR$LOW$SCREEN;
867      END; /* ELSE */
      ENABLE;
868      END; /* IF TEST$CHAR = 'P' */
      ELSE
869      IF TEST$CHAR = 'G' THEN
870      DO;
871      DISABLE;
872      CALL SET$LOW$HOME;
873      CALL SEND$STRING$CRT(.
874
875
```



```

$' ));
876      4
877      4
878      5
879      5
STATUS);
880      5
881      5
882      5
STATUS);
883      5
884      5
885      4
886      4
887      4
888      4

889      3
890      3
891      4
892      4
893      4

EN (Y/N) $' ));

('GENERAL DATA STATISTICS TO PRINTER (Y/N)

IF CHECK$YES$NO THEN
DO;
CALL INITIALIZE$PRINTER;
CALL CONSOL(.(':CI:$'),.(':TO:$'),..CONSOL$
STATUS);
CALL LOAD$TIME;
CALL WRITE$BIGPICTURE$LP;
CALL CONSOL(.(':CI:$'),.(':VO:$'),..CONSOL$
STATUS);
CALL CLOSE$PRINTER;
END; /* IF CHECK$INPUT */
ELSE CALL CLOCK$TIMER (2);
CALL CLEAR$LOW$SCREEN;

ENABLE;
END;
ELSE
IF TEST$CHAR='B' THEN
DO;
DISABLE;
CALL SET$LOW$HOME;
CALL SEND$STRING$CRT(.
('GENERAL DATA STATISTICS TO SCREE
```



```
894 4 IF CHECK$YES$NO THEN
895 4 DO;
896 5 CALL CLOCK$TIMER(2);
897 5 CALL INITIALIZE$SCREEN;
898 5 CALL LOAD$TIME;
899 5 CALL WRITE$BIGPICTURE;
900 5 END; /* IF CHECK$INPUT */
901 4 ELSE CALL CLOCK$TIMER(2);
902 4 CALL CLEAR$LOW$SCREEN;
903 4 ENABLE;
904 4 END;
ELSE
905 3 IF TEST$CHAR='T' THEN
906 3 DO;
907 4 DISABLE;
908 4 CALL SET$LOW$HOME;
909 4 CALL SEND$STRING$CRT(
; 5 ('ENTER NEW DATE/TIME (Y/N) $'))
910 4 IF CHECK$YES$NO THEN
911 4 DO;
912 5 CALL CLOCK$TIMER(2);
```



```
913      CALL CLEAR$LOW$SCREEN; 5
914      CALL SET$LOW$HOME; 5
915      CALL INITIATE$TIME; 5
916      END; /* IF CHECK$INPUT */ 5
917      ELSE CALL CLOCK$TIMER(2); 4
918      CALL CLEAR$LOW$SCREEN; 4
919      ENABLE; 4
920      END; 4

921      ELSE 3
922      IF TEST$CHAR='W' THEN 3
923      DO; 4
924      DISABLE; 4
925      CALL SET$LOW$HOME; 4
926      CALL LOAD$TIME; 4
927      CALL WRITE(0,.M0(0),73,.WRITE$STATUS) 4
928      DO XY = 1 TO 180; 4
929      CALL TIME(250); 5
930      END; /* DO XY */ 5
931      CALL CLEAR$LOW$SCREEN; 4
932      ENABLE; 4
933      END; 4
```



```

ELSE
    933 IF TEST$CHAR='R' THEN
    934 DO;
    935 DISABLE;
    936 CALL SET$LOW$HOME;
    937 CALL SEND$STRING$CRT(.( 'VERIFY DATA
RECEPTION.(Y/N
) $' ));
    938
    939 IF CHECK$YES$NO THEN RECEIVE$PRINT=T
    ELSE RECEIVE$PRINT=FALSE;
    940 CALL CLOCK$TIMER(2);
    941 CALL CLEAR$LOW$SCREEN;
    942
    943 ENABLE;
    944 END; /* 'R' */
    ELSE
    945 IF TEST$CHAR='S' THEN
    946 DO;
    947 DISABLE;
    948 CALL SET$LOW$HOME;
    949 CALL SEND$STRING$CRT(.(
    ('PRINT GENERAL DATA STATISTICS PER D
    (Y/N) $' ));
ATA RECEPTION
    -

```



```
950 4 IF CHECK$YES$NO THEN SCREEN$PRINT=TR
UE;
952 4 ELSE SCREEN$PRINT=FALSE;
953 4 CALL CLOCK$TIMER(2);
954 4 CALL CLEAR$LOW$SCREEN;
955 4 ENABLE;
956 4 END ; /* IF 'S' */
ELSE
957 3 IF TEST$CHAR = 'Z' THEN
958 3 DO;
959 4 DISABLE;
960 4 CALL SET$LOW$HOME;
961 4 CALL SEND$STRING$CRT(.
( 'REASSIGN PLASMA DEVICES (Y/N)
$' ));
962 4 IF CHECK$YES$NO THEN
963 4 DO;
964 5 CALL CLOCK$TIMER(2);
965 5 OK = 0;
966 5 DO WHILE OK = 0;
967 6 CALL CLEAR$LOW$SCREEN;
968 6 CALL SET$LOW$HOME;
969 6 CALL SEND$STRING$CRT(.

```



```
TCH (#) '$');
970 6
971 6
LY '$');
972 6

0 (3)PLASMA.
- ONE$');
973 6
MA.TWO '$');
974 6
975 6

3. '$');
976 6
977 6
978 6
'3');
979 7

DATA AGAIN.$' )
- );
980 7
981 7
982 7
983 7
984 6

('OUTPUT GRAPHIC DISPLAY TO SWI
CALL CRLF;
CALL SEND$STRING$CRT(.( '(0)LP ON
CALL SEND$STRING$CRT(
( '(1)PLASMA.ONE (2)PLASMA.TW

CALL SEND$STRING$CRT(.( ' AND PLAS
CALL CRLF;
CALL SEND$STRING$CRT(
( 'INPUT SWITCH NUMBER: 0,1,2 OR
XY1 = READ$CHAR$CRT;
CALL SEND$CHAR$CRT(XY1);
DO WHILE ((XY1 < '0') OR (XY1 >
CALL SEND$STRING$CRT(
( ' INPUT DATA INCORRECT,ENTER

XY1 = READ$CHAR$CRT;
CALL BACK$SPACE$ERASE(40);
CALL SEND$CHAR$CRT(XY1);
END; /* WHILE 1<X>3 */
OK = CHECK$INPUT;
```



```
985 6 IF OK THEN PLASMA$SWITCH = (XY1  
- '0.');
```

```
987 6 END; /* WHILE OK */  
988 5 END; /* IF CHECK$YES$NO */  
989 4 CALL CLOCK$TIMER(2);  
990 4 CALL CLEAR$LOW$SCREEN;  
991 4 ENABLE;  
992 4 END; /* IF TEST$CHAR = 'Z' */  
ELSE  
993 3 IF TEST$CHAR = 'M' THEN  
994 3 DO;  
995 4 DISABLE;  
996 4 CALL SET$LOW$HOME;  
997 4 CALL SEND$STRING$CRT(.  
; ('LIST COMMAND OPTIONS: (Y/N) $'))  
IF CHECK$YES$NO THEN  
998 4 DO;  
999 4 CALL CLOCK$TIMER(2);  
1000 5 CALL INITIALIZE$SCREEN;  
1001 5 CALL LIST$MENU$COMMANDS;  
1002 5 END; /* IF CHECK$YES$NO */  
1003 5 ELSE
```



```

1004 4 DO;
1005 5 CALL CLOCK$TIMER(2);
1006 5 CALL CLEAR$LOW$SCREEN;
1007 5 END; /* ELSE */
1008 4 ENABLE;
1009 4 END; /* IF 'M' */
      ELSE
1010 3 IF TEST$CHAR = 'N' THEN
1011 3 DO;
1012 4 DISABLE;
1013 4 CALL SET$LOW$HOME;
1014 4 CALL SEND$STRING$CRT(
      ('LIST COMMAND OPTIONS TO LINE P
RINTER (Y/N) $
-
'));
1015 4 IF CHECK$YES$NO THEN
1016 4 DO;
1017 5 CALL CLOCK$TIMER(2);
1018 5 CALL INITIALIZE$PRINTER;
1019 5 CALL CONSOL((':CI:$'), (':TO:$'
), .CONSOL$STAT
-
US);
1020 5 DO XY1 = 1 TO 15;

```



```

1021      6      CALL PCRLF;
1022      6      END;
1023      5      CALL LOAD$TIME;
1024      5      CALL WRITE(0,.M0(0),73,.WRITE$ST
ATUS);
1025      5      DO XY1 = 1 TO 5;
1026      6      CALL PCRLF;
1027      6      END;
1028      5      CALL LIST$MENU$COMMANDS$LP;
1029      5      CALL CONSOL((':CI:$'),.(:VO:$'
),.CONSOL$STAT
-
US);
1030      5      CALL CLOSE$PRINTER;
1031      5      CALL CLEAR$LOW$SCREEN;
1032      5      END; /* IF CHECK$YES$NO */
1033      4      ELSE
DO;
1034      5      CALL CLOCK$TIMER(2);
1035      5      CALL CLEAR$LOW$SCREEN;
1036      5      END; /* ELSE */
1037      4      ENABLE;
1038      4      END; /* IF 'N' */
END; /* IF ROR INPUT(247) */

```


1040 2 END;
1041 1 END; /* SYSTEM MODULE */

MODULE INFORMATION:

CODE AREA SIZE	= 1FF8H	8184D
VARIABLE AREA SIZE	= 016AH	362D
MAXIMUM STACK SIZE	= 0010H	16D
1499 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SCREENMODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:SCREEN.MOD NOOBJECT PAGEWIDTH(82) PA
GELENGTH(24) D
-ATE(MAY 1979) TITLE('MODULE SCREEN.MOD')

```
1      SCREEN$MODULE:
      DO;
2      1      DECLARE LIT LITERALLY 'LITERALLY';
3      1      DECLARE DCL LIT 'DECLARE';

4      1      DCL (IDA, IDB, TEST$CHAR) BYTE;
5      1      DCL XY ADDRESS;
6      1      DCL ERASE$COUNT ADDRESS PUBLIC;
7      1      DCL CONVERTED$BYTE$NUMBER (5) BYTE PUBLIC ;
8      1      DCL WRITE$STATUS ADDRESS;
```

```
/* EXTENALS FOR TIME PARAMETERS */
9      1      DCL (MILI$SEC, DUMMY$SEC, SECONDS, MINUTES, HOURS, DAY, MONT
H, YEAR, SEC$TIM
```



```

- E)
10 1      BYTE EXTERNAL;
11 1      DCL TIME$STEP ADDRESS EXTERNAL;
11 1      DCL TIME$BUFFER(6) BYTE EXTERNAL;

```

\$NOLIST

```

218 1      DCL L0 (*) BYTE INITIAL
D | BEARING | ( | CONTACT | QUADRANT | STATUS | TYPE | COURSE | SPEED
- |-----|
219 1      DCL L1 (*) BYTE INITIAL
- |-----|
220 1      DCL L2 (*) BYTE INITIAL
S | DG | YD |
- |-----|

```


236 1

DCL M0 (73) BYTE PUBLIC;

237 1

DCL M1 (*) BYTE INITIAL

(' ;

ISTICS

-

;

STATISTICS ;

OWNSHIP STAT

238 1

DCL M2 (*) BYTE INITIAL

(' ;

239

1

DCL M3 (*) BYTE INITIAL

(' ;

COURSE-----

DGS ;

COURSE----

240 1

DCL M4 (*) BYTE INITIAL

(' ;

SPEED-----

KTS ;

SPEED----

241 1

DCL M5 (*) BYTE INITIAL

(' ;

BEARING-----

DGS ;

LATITUDE----


```

242 1          DCL M6 (*) BYTE INITIAL
-----:      ('| RANGE-----:
              . - DGS |
              ');
243 1          DCL M7 (*) BYTE INITIAL
              ('| CPA TIME-----:
UMBER:        - |
              ');
244 1          DCL M8 (*) BYTE INITIAL
              ('| CPA DISTANCE-----:
              - |
              ');
245 1          DCL M9 (*) BYTE INITIAL
              ('| LATITUDE-----:
              - |
              ');
246 1          DCL M10 (*) BYTE INITIAL
              ('| LONGITUDE-----:
              - |
              ');
  
```

YDS | LONGITUDE-

H.M | QUADRANT N

YDS |

DGS |

DGS |

247 1 DCL M11 (*) BYTE INITIAL
(' ' COLLISION STATUS: ;

- ;

248 1 DCL M12 (*) BYTE INITIAL
(' ' QUADRANT NUMBER-: ;

- ;

249 1 DCL M13 (*) BYTE INITIAL
(' '-----|-----

- -----|

);

250 1 WRITE\$BIGPICTURE: PROCEDURE PUBLIC;

/* THIS PROCEDURE WRITES BIG\$PICTURE TO MDS CRT OR LI

NE PRINTER */

251 2 DCL (I,J,K) ADDRESS ;

252 2 I=73;

253 2 CALL WRITE (0,.M0(0),I,.WRITE\$STATUS);

254	2	CALL CRLF;	
255	2	CALL WRITE	(0,.L0(0),I,..WRITE\$STATUS);
256	2	CALL CRLF;	
257	2	CALL WRITE	(0,.L1(0),I,..WRITE\$STATUS);
258	2	CALL CRLF;	
259	2	CALL WRITE	(0,.L2(0),I,..WRITE\$STATUS);
260	2	CALL CRLF;	
261	2	CALL WRITE	(0,.L3(0),I,..WRITE\$STATUS);
262	2	CALL CRLF;	
263	2	CALL WRITE	(0,.L4(0),I,..WRITE\$STATUS);
264	2	CALL CRLF;	
265	2	CALL WRITE	(0,.L5(0),I,..WRITE\$STATUS);
266	2	CALL CRLF;	
267	2	CALL WRITE	(0,.L6(0),I,..WRITE\$STATUS);
268	2	CALL CRLF;	
269	2	CALL WRITE	(0,.L7(0),I,..WRITE\$STATUS);
270	2	CALL CRLF;	
271	2	CALL WRITE	(0,.L8(0),I,..WRITE\$STATUS);
272	2	CALL CRLF;	
273	2	CALL WRITE	(0,.L9(0),I,..WRITE\$STATUS);
274	2	CALL CRLF;	


```

275 2 CALL WRITE (0,.L10(0),I,.WRITE$STATUS);
276 2 CALL CRLF;
277 2 CALL WRITE (0,.L11(0),I,.WRITE$STATUS);
278 2 CALL CRLF;
279 2 CALL WRITE (0,.L12(0),I,.WRITE$STATUS);
280 2 CALL CRLF;
281 2 CALL WRITE (0,.L13(0),I,.WRITE$STATUS);
282 2 CALL CRLF;
283 2 CALL WRITE (0,.L14(0),I,.WRITE$STATUS);
284 2 CALL CRLF;
285 2 CALL WRITE (0,.L15(0),I,.WRITE$STATUS);
286 2 CALL CRLF;
287 2 CALL WRITE (0,.L16(0),I,.WRITE$STATUS);
288 2 CALL CRLF;
289 2 CALL WRITE (0,.L17(0),I,.WRITE$STATUS);
290 2 CALL CRLF;
291 2 END; /* WRITE$BIGPICTURE */

292 1 WRITE$LITTLEPICTURE: PROCEDURE PUBLIC;
```



```
LINE PRINTER      /* THIS PROCEDURE WRITES LITTLE$PICTURE TO MDS CRT OR
-                */
293 2    DCL (L,M,N) ADDRESS;
294 2    L=73;
295 2    CALL WRITE(0,.M0(0),L,..WRITE$STATUS);
296 2    CALL CRLF;
297 2    CALL WRITE(0,.M1(0),L,..WRITE$STATUS);
298 2    CALL CRLF;
299 2    CALL WRITE(0,.M2(0),L,..WRITE$STATUS);
300 2    CALL CRLF;
301 2    CALL WRITE(0,.M3(0),L,..WRITE$STATUS);
302 2    CALL CRLF;
303 2    CALL WRITE(0,.M4(0),L,..WRITE$STATUS);
304 2    CALL CRLF;
305 2    CALL WRITE(0,.M5(0),L,..WRITE$STATUS);
306 2    CALL CRLF;
307 2    CALL WRITE(0,.M6(0),L,..WRITE$STATUS);
308 2    CALL CRLF;
309 2    CALL WRITE(0,.M7(0),L,..WRITE$STATUS);
310 2    CALL CRLF;
```


MAY 1

```

311      2      CALL WRITE(0,.M8(0),L,.WRITE$STATUS);
312      2      CALL CRLF;
313      2      CALL WRITE(0,.M9(0),L,.WRITE$STATUS);
314      2      CALL CRLF;
315      2      CALL WRITE(0,.M10(0),L,.WRITE$STATUS);
316      2      CALL CRLF;
317      2      CALL WRITE(0,.M11(0),L,.WRITE$STATUS);
318      2      CALL CRLF;
319      2      CALL WRITE(0,.M12(0),L,.WRITE$STATUS);
320      2      CALL CRLF;
321      2      CALL WRITE(0,.M13(0),L,.WRITE$STATUS);
322      2      CALL CRLF;
323      2      END; /*WRITE$LITTLEPICTURE */

```



```

TO LP. */
324 1  =
      =
      =
325 2  =
326 2  =
      =
      =
      =
327 1  =
      =
      =
328 2  =
329 2  =
330 3  =
331 2  =
332 2  =
      =
      =
      =

```

```

/* INCLUDE PROCEDURES TO WRITE BIG AND LITTLE PICTURE
$INCLUDE(:F1:WRITE.LP)
OUTPUT$STATUS$LP: PROCEDURE BYTE;

/* TRUE IF DATA OUTPUT LINE TO LP READY */
RETURN ROR(INPUT(245),2);
END;

SEND$CHAR$LP: PROCEDURE (CHAR);

/* SEND A CHARACTER TO THE LINE PRINTER */
DCL CHAR BYTE;
DO WHILE NOT OUTPUT$STATUS$LP;
  END; /* DO WHILE */
OUTPUT(244)=CHAR;
END;

```



```

333 1 = LPCRLF: PROCEDURE;
    =
    = /* SEND CARRIAGE RETURN AND LINE FEED TO THE LP. */
334 2 = CALL SEND$CHAR$LP(0DH);
335 2 = CALL SEND$CHAR$LP(0AH);
336 2 = END;
    =
    =
    =
    =
337 1 = WRITE$BIGPICTURE$LP: PROCEDURE PUBLIC;
    =
    = /* THIS PROCEDURE WRITES BIG$PICTURE TO MDS CRT OR LI
NE PRINTER */
338 2 = DCL (I,J,K) ADDRESS ;
339 2 = I=73;
340 2 = DO J = 1 TO 15 ;
341 3 =     CALL LPCRLF;
342 3 =     END; /* DO J */
343 2 =     CALL WRITE (0,.M0(0),I,.WRITE$STATUS);
344 2 =     CALL LPCRLF;
345 2 =     DO J= 1 TO 5;

```



```
346 CALL LPCRLF;  
347 END; /* DO J */  
348 CALL WRITE (0,.L17(0),I,.WRITE$STATUS);  
349 CALL LPCRLF;  
350 CALL WRITE (0,.L0(0) ,I,.WRITE$STATUS);  
351 CALL LPCRLF;  
352 CALL WRITE (0,.L1(0),I,.WRITE$STATUS);  
353 CALL LPCRLF;  
354 CALL WRITE (0,.L2(0),I,.WRITE$STATUS);  
355 CALL LPCRLF;  
356 CALL WRITE (0,.L3(0),I,.WRITE$STATUS);  
357 CALL LPCRLF;  
358 CALL WRITE (0,.L4(0),I,.WRITE$STATUS);  
359 CALL LPCRLF;  
360 CALL WRITE (0,.L5(0),I,.WRITE$STATUS);  
361 CALL LPCRLF;  
362 CALL WRITE (0,.L6(0),I,.WRITE$STATUS);  
363 CALL LPCRLF;  
364 CALL WRITE (0,.L7(0),I,.WRITE$STATUS);  
365 CALL LPCRLF;  
366 CALL WRITE (0,.L8(0),I,.WRITE$STATUS);
```



```
367      CALL LPCRLF;
368      CALL WRITE (0,.L9(0),I,..WRITE$STATUS);
369      CALL LPCRLF;
370      CALL WRITE (0,.L10(0),I,..WRITE$STATUS);
371      CALL LPCRLF;
372      CALL WRITE (0,.L11(0),I,..WRITE$STATUS);
373      CALL LPCRLF;
374      CALL WRITE (0,.L12(0),I,..WRITE$STATUS);
375      CALL LPCRLF;
376      CALL WRITE (0,.L13(0),I,..WRITE$STATUS);
377      CALL LPCRLF;
378      CALL WRITE (0,.L14(0),I,..WRITE$STATUS);
379      CALL LPCRLF;
380      CALL WRITE (0,.L15(0),I,..WRITE$STATUS);
381      CALL LPCRLF;
382      CALL WRITE (0,.L16(0),I,..WRITE$STATUS);
383      CALL LPCRLF;
384      CALL WRITE (0,.L17(0),I,..WRITE$STATUS);
385      CALL LPCRLF;
386      END; /* WRITE$BIGPICTURE */
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
=
```



```

=
=
387 1 = WRITE$LITTLEPICTURE$LP: PROCEDURE PUBLIC;
=
=
LINE PRINTER
=
388 2 = DCL (L,M,N) ADDRESS;
389 2 = L=73;
390 2 = DO M = 1 TO 15;
391 3 = CALL LPCRLF;
392 3 = END; /* DO M */
393 2 = CALL WRITE(0,.M0(0),L,.WRITE$STATUS);
394 2 = CALL LPCRLF;
395 2 = DO M = 1 TO 5;
396 3 = CALL LPCRLF;
397 3 = END; /* DO M */
398 2 = CALL WRITE(0,.M13(0),L,.WRITE$STATUS);
399 2 = CALL LPCRLF;
400 2 = CALL WRITE(0,.M1(0),L,.WRITE$STATUS);
401 2 = CALL LPCRLF;
402 2 = CALL WRITE(0,.M2(0),L,.WRITE$STATUS);

```



```
403      = CALL LPCRLF;  
404      = CALL WRITE(0,.M3(0),L,.WRITE$STATUS);  
405      = CALL LPCRLF;  
406      = CALL WRITE(0,.M4(0),L,.WRITE$STATUS);  
407      = CALL LPCRLF;  
408      = CALL WRITE(0,.M5(0),L,.WRITE$STATUS);  
409      = CALL LPCRLF;  
410      = CALL WRITE(0,.M6(0),L,.WRITE$STATUS);  
411      = CALL LPCRLF;  
412      = CALL WRITE(0,.M7(0),L,.WRITE$STATUS);  
413      = CALL LPCRLF;  
414      = CALL WRITE(0,.M8(0),L,.WRITE$STATUS);  
415      = CALL LPCRLF;  
416      = CALL WRITE(0,.M9(0),L,.WRITE$STATUS);  
417      = CALL LPCRLF;  
418      = CALL WRITE(0,.M10(0),L,.WRITE$STATUS);  
419      = CALL LPCRLF;  
420      = CALL WRITE(0,.M11(0),L,.WRITE$STATUS);  
421      = CALL LPCRLF;  
422      = CALL WRITE(0,.M12(0),L,.WRITE$STATUS);  
423      = CALL LPCRLF;
```



```

424 2 = CALL WRITE(0,.M13(0),L,.WRITE$STATUS);
425 2 = CALL LPCRLF;
426 2 = END; /*WRITE$LITTLEPICTURE */

427 1 CONVERT$ADDRESS$TO$CHARS: PROCEDURE (CHAR$ADDRESS) PUB
LIC;

RESS
*/
428 2 DCL SCALE$FACTOR (5) ADDRESS DATA (10000,1000,100,10,
1);
429 2 DCL (JA,IA,PASS$COUNT) BYTE ;
430 2 DCL (CHAR,TEMP,CHAR$ADDRESS) ADDRESS;
431 2 CHAR=CHAR$ADDRESS;
432 2 JA=4;
433 2 DO IA=0 TO LAST(SCALE$FACTOR);
434 3 PASS$COUNT=0;
435 3 TEMP=SCALE$FACTOR(IA);
436 3 DO WHILE CHAR >= TEMP;
437 4 CHAR=CHAR-TEMP;
438 4 PASS$COUNT=PASS$COUNT+1;
439 4 END; /* DO WHILE CHAR */

```



```

440      3      CONVERTED$BYTE$NUMBER(JA)=(PASS$COUNT+'0');
441      3      JA=JA-1;
442      3      END; /* DO IA */
443      2      END; /* CONVERT$ADDRESS$TO$CHARS */

444      1      CONVERT$BYTE$TO$CHARS:PROCEDURE (CHAR$BYTE) PUBLIC ;

/* THIS PROCEDURE CONVERTS THE NUMERIC VALUE OF A BYT

      TO 3 CHARACTER BYTE REPRESENTATION . */
      DCL SCALE$FACTOR (3) BYTE DATA (100,10,1) ;
      DCL (JB,IB,CHAR,CHAR$BYTE,PASS$COUNT,TEMP) BYTE;
      CHAR=CHAR$BYTE;
      JB=2;
      DO IB=0 TO LAST(SCALE$FACTOR);
          PASS$COUNT=0;
          TEMP=SCALE$FACTOR(IB);
          DO WHILE CHAR>= TEMP;
              CHAR=CHAR-TEMP;
              PASS$COUNT=PASS$COUNT + 1;
          
```

E VARIABLE

```

445      2
446      2
447      2
448      2
449      2
450      3
451      3
452      3
453      4
454      4

```



```

455 4      END; /* DO WHILE */
456 3      CONVERT$BYTE$NUMBER(JB) = ( PASS$COUNT + '0' );
457 3      JB=JB-1;
458 3      END; /* DO IB */
459 2      END; /* CONVERT$BYTE$TO$CHARS$ */

460 1      LOAD$DATA$BIGPICTURE: PROCEDURE (PTR,P$COUNT,IDATA,JDA
TA);

E TO

PRINTER */
461 2      DCL (PTR,P$COUNT,TEMP) ADDRESS;
462 2      DCL (I,J,IDATA,JDATA) BYTE; /* IDATA=INDEX , JDATA=SH
IPINDEX. */
463 2      DCL P BASED PTR BYTE ;
464 2      DCL DIRECTION BYTE;
465 2      I=IDATA;
466 2      J=JDATA;
/* INITIALIZE CONTACT NUMBER */
467 2      PTR =PTR+4;
468 2      DO CASE I;
469 3      P='F';

```



```

470      P='H';
471      P='U';
472      END; /* CASE */
473      PTR=PTR+1;
474      P=(J+'0');
      /* INITIALIZE QUADRANT */
475      PTR=PTR+8;
476      CALL CONVERT$BYTE$TO$CHARS(SHIP$PLOT(I).QUADRANT(J));
477      P=CONVERTED$BYTE$NUMBER(1);
478      PTR=PTR+1;
479      P=CONVERTED$BYTE$NUMBER(0);
      /* INITIALIZE STATUS */
480      PTR=PTR+8;
481      DO CASE I;
482      P='F';
483      P='H';
484      P='U';
485      END; /* CASE */
486      PTR=PTR+1;
487      DO CASE I;
488      P='R';

```



```

489 P='O';
490 P='N';
491 END; /* CASE */
      /* INITIALIZE TYPE *****/
492 PTR=PTR+8;
493 PTR=PTR+1;
      /* INITIALIZE COURSE */
494 PTR=PTR+6;
495 CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).COURSE(J));
496 P=CONVERTED$BYTE$NUMBER(2);
497 PTR=PTR+1;
498 P=CONVERTED$BYTE$NUMBER(1);
499 PTR=PTR+1;
500 P=CONVERTED$BYTE$NUMBER(0);
      /* INITIALIZE SPEED */
501 PTR=PTR+7;
502 CALL CONVERT$BYTE$TO$CHARS(SHIP$PLOT(I).SPEED(J));
503 P=CONVERTED$BYTE$NUMBER(1);
504 PTR=PTR+1;
505 P=CONVERTED$BYTE$NUMBER(0);
      /* INITIALIZE BEARING */

```



```

506           /* CHECK FLAG FOR PORT OR STARBOARD */
507       PTR=PTR+7;
508       IF SHIP$PLOT(I).BEARING(J) >= 02000H THEN
509       DO;
510       DIRECTION = 'S';
511       TEMP = SHIP$PLOT(I).BEARING(J) AND 01FFFFH;
              END;
              ELSE
512       DO;
513       DIRECTION='P';
514       TEMP = SHIP$PLOT(I).BEARING(J);
515       END;
516       CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
517       P=CONVERTED$BYTE$NUMBER(2);
518       PTR=PTR+1;
519       P=CONVERTED$BYTE$NUMBER(1);
520       PTR=PTR+1;
521       P=CONVERTED$BYTE$NUMBER(0);
522       PTR = PTR + 1;
523       P=DIRECTION;
524       IF ((I=0) AND (J=1)) THEN P=' ';
```



```

526 2 IF ((TEMP = 0) OR (TEMP = 180)) THEN P = ' ';
    /* INITIALIZE RANGE */
528 2 PTR=PTR+6;
529 2 CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).RANGE(J));
530 2 P=CONVERTED$BYTE$NUMBER(4);
531 2 PTR=PTR+1;
532 2 P=CONVERTED$BYTE$NUMBER(3);
533 2 PTR=PTR+1;
534 2 P=CONVERTED$BYTE$NUMBER(2);
535 2 PTR=PTR+1;
536 2 P=CONVERTED$BYTE$NUMBER(1);
537 2 PTR=PTR+1;
538 2 P=CONVERTED$BYTE$NUMBER(0);
539 2 PTR=PTR+1;
540 2 P='0';
541 2 END; /* LOAD$DATA$BIGPICTURE */

542 1 LOAD$DATA$LITTLEPICTURE: PROCEDURE (ILITTLE,JLITTLE) P
PUBLIC ;

```



```

Y/LINE      /* THIS PROCEDURE LOADS A SPECIFIC CONTACT INTO MEMOR
543      2      ARRAY FOR MAPPING TO THE CRT OR LINE PRINTER. */
          DCL (I,J,ILITTLE,JLITTLE) BYTE ; /* ILITTLE=INDEX , JL
ITITLE=SHIPINDE
          X. */
544      2      DCL (N,K) BYTE;
545      2      DCL DIRECTION BYTE;
546      2      DCL TEMP ADDRESS;
547      2      I=ILITTLE;
548      2      J=JLITTLE;
          /* INITIALIZE CONTACT NUMBER */
549      2      M1(20)=I;
550      2      M1(21)=J;
          /* CONVERT F,U,H SYMBOLS TO SHIP PLOT INDEX . */
551      2      IF I='F' THEN I=0;
          ELSE
553      2      IF I='H' THEN I=1;
          ELSE
555      2      IF I='U' THEN I=2;
          J=J-'0';
          /* INITIALIZE CONTACT AND OWNERSHIP COURSE */
558      2      CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).COURSE(J));

```



```

559      N=26;
560      DO K=0 TO 2;
561          M3(N-K)=CONVERTED$BYTE$NUMBER(K);
562      END;
563      CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(0).COURSE(1));
564      N=64;
565      DO K=0 TO 2;
566          M3(N-K)=CONVERTED$BYTE$NUMBER(K);
567      END;
568      /* INITIALIZE CONTACT AND OWN SHIP SPEED */
569      CALL CONVERT$BYTE$TO$CHARS(SHIP$PLOT(1).SPEED(J));
570      N=26;
571      DO K=0 TO 1;
572          M4(N-K)=CONVERTED$BYTE$NUMBER(K);
573      END;
574      CALL CONVERT$BYTE$TO$CHARS(SHIP$PLOT(0).SPEED(1));
575      N=64;
576      DO K=0 TO 1;
577          M4(N-K)=CONVERTED$BYTE$NUMBER(K);
578      END;
579      /* INITIALIZE CONTACT BEARING AND OWN SHIP LATITUDE */

```



```

578 2      /* CHECK FLAG FOR PORT OR STARBOARD */
579 2      IF SHIP$PLOT(I).BEARING(J) >= 02000H THEN
580 3          DO;
581 3              DIRECTION='S';
582 3              TEMP = SHIP$PLOT(I).BEARING(J) AND 01FFFH;
            END;
            ELSE
583 2          DO;
584 3              DIRECTION = 'P';
585 3              TEMP = SHIP$PLOT(I).BEARING(J);
586 3              END;
587 2              CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
588 2              N=25;
589 2              DO K = 0 TO 2;
590 3                  M5(N-K)=CONVERTED$BYTE$NUMBER(K);
591 3              END;
592 2              M5(26)= DIRECTION ;
593 2              IF ((I=0) AND (J=1)) THEN M5(26)=' ' ;
594 2              IF ((TEMP = 0) OR (TEMP = 180)) THEN M5(26) = ' ' ;
595 2              /* CHECK FLAG FOR NORTH OR SOUTH */
597 2              IF SHIP$PLOT(0).LAT(1) >= 02000H THEN

```



```
598 DO;
599 M5(64)='N';
600 TEMP = SHIP$PLOT(0).LAT(1) AND 01FFFFH;
601 END;
    ELSE
602 DO;
603 M5(64)='S';
604 TEMP = SHIP$PLOT(0).LAT(1);
605 END;
606 CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
607 M5(60)=CONVERTED$BYTE$NUMBER(2);
608 M5(61)=CONVERTED$BYTE$NUMBER(1);
609 M5(63)=CONVERTED$BYTE$NUMBER(0);
    /* INITIALIZE CONTACT RANGE AND OWNERSHIP LONGITUDE */
610 CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).RANGE(J));
611 M6(26)='0';
612 N=25;
613 DO K=0 TO 4;
614 M6(N-K)=CONVERTED$BYTE$NUMBER(K);
615 END;
    /* CHECK FLAG FOR WEST OR EAST */
```



```

616 2 IF SHIP$PLOT(0).LONG(1) >= 02000H THEN
617 2 DO;
618 3 M6(64)='W';
619 3 TEMP = SHIP$PLOT(0).LONG(1) AND 01FFFH;
620 3 END;
    ELSE
621 2 DO;
622 3 M6(64)='E';
623 3 TEMP = SHIP$PLOT(0).LONG(1);
624 3 END;
625 2 CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
626 2 N=63;
627 2 DO K=0 TO 3;
628 3 IF K=1 THEN N=N-1;
629 3 M6(N-K)=CONVERT$BYTE$NUMBER(K);
630 3 END;
631 3 /*INITIALIZE CONTACT CPA-TIME AND OWNERSHIP QUADRANT NUM
BER */
632 2 CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).CPA$TIME(J)
);
633 2 N=26;
634 2 DO K=0 TO 3;
635 3 IF K=2 THEN N=N-1;

```



```

637      M7(N-K)=CONVERTED$BYTE$NUMBER(K);
638      END;
639      CALL CONVERT$BYTE$TO$CHARS(SHIP$PLOT(0).QUADRANT(1));
640      M7(63)=CONVERTED$BYTE$NUMBER(1);
641      M7(64)=CONVERTED$BYTE$NUMBER(0);
        /* INITIALIZE CONTACT CPA-DISTANCE */
642      CALL CONVERT$ADDRESS$TO$CHARS(SHIP$PLOT(I).CPA$DISTANC
E(J));
643      N=25;
644      DO K=0 TO 4;
645          M8(N-K)=CONVERTED$BYTE$NUMBER(K);
646      END;
647      M8(26)='0';
        /* INITIALIZE CONTACT LATITUDE */
        /* CHECK FLAG FOR NORTH OR SOUTH */
        IF SHIP$PLOT(I).LAT(J) >= 02000H THEN
            DO;
                M9(26)='N';
                TEMP = SHIP$PLOT(I).LAT(J) AND 01FFFH;
            END;
        ELSE
            DO;
653

```



```

654      M9(26)='S';
655      TEMP = SHIP$PLOT(I).LAT(J);
656      END;
657      CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
658      N=25;
659      DO K=0 TO 2;
660          IF K=1 THEN N=N-1;
661          M9(N-K)=CONVERTED$BYTE$NUMBER(K);
662      END;
663      /* INITIALIZE CONTACT LONGITUDE */
664      /* CHECK FLAG FOR WEST OR EAST */
665      IF SHIP$PLOT(I).LONG(J) >= 02000H THEN
666          DO;
667              M10(26)='W';
668              TEMP = SHIP$PLOT(I).LONG(J) AND 01FFFH;
669          ELSE
670              DO;
671                  M10(26)='E';
672                  TEMP = SHIP$PLOT(I).LONG(J);
673              END;
674      END;

```



```

673 2 CALL CONVERT$ADDRESS$TO$CHARS(TEMP);
674 2 N=25;
675 2 DO K=0 TO 3;
676 3 IF K=1 THEN N=N-1;
678 3 M10(N-K)=CONVERTED$BYTE$NUMBER(K);
679 3 END;
/*INITIALIZE CONTACT COLLISION STATUS */
680 2 IF SHIP$PLOT(I).COLLISION$FLAG(J)=03H THEN
681 2 DO;
682 3 M11(20)=0EH;
683 3 M11(30)=18H;
684 3 M11(21)='C';
685 3 M11(22)='O';
686 3 M11(23)='L';
687 3 M11(24)='L';
688 3 M11(25)='I';
689 3 M11(26)='S';
690 3 M11(27)='I';
691 3 M11(28)='O';
692 3 M11(29)='N';
693 3 END; /* IF */

```



```

694      ELSE
695      DO;
696      N=20;
697      DO K=0 TO 10;
698      M11(N)=',';
699      N=N+1;
700      END; /* DO */
        END; /* ELSE */
        /* INITIALIZE CONTACT QUADRANT NUMBER */
        CALL CONVERT$BYTES TO$CHARS(SHIP$PLOT(I).QUADRANT(J));
        M12(25)=CONVERTED$BYTE$NUMBER(1);
        M12(26)=CONVERTED$BYTE$NUMBER(0);
        END; /* LOAD$DATA$LITTLEPICTURE */

705      1      LOAD$LINE$ARRAY: PROCEDURE (A,B,C) ;

```


/* THIS PROCEDURE PASSES LINE ARRAY, INDEX, AND SHIPIND

EX TO

LOAD DATA TO BIG OR LITTLE PICTURE. */

DCL (A,D) ADDRESS;

DCL (B,C) BYTE ;

D=73;

DO CASE A;

706	2	CALL LOAD\$DATA\$BIGPICTURE(.L0(0),D,B,C);
707	2	CALL LOAD\$DATA\$BIGPICTURE(.L1(0),D,B,C);
708	2	CALL LOAD\$DATA\$BIGPICTURE(.L2(0),D,B,C);
709	2	CALL LOAD\$DATA\$BIGPICTURE(.L3(0),D,B,C);
710	3	CALL LOAD\$DATA\$BIGPICTURE(.L4(0),D,B,C);
711	3	CALL LOAD\$DATA\$BIGPICTURE(.L5(0),D,B,C);
712	3	CALL LOAD\$DATA\$BIGPICTURE(.L6(0),D,B,C);
713	3	CALL LOAD\$DATA\$BIGPICTURE(.L7(0),D,B,C);
714	3	CALL LOAD\$DATA\$BIGPICTURE(.L8(0),D,B,C);
715	3	CALL LOAD\$DATA\$BIGPICTURE(.L9(0),D,B,C);
716	3	CALL LOAD\$DATA\$BIGPICTURE(.L10(0),D,B,C);
717	3	CALL LOAD\$DATA\$BIGPICTURE(.L11(0),D,B,C);
718	3	CALL LOAD\$DATA\$BIGPICTURE(.L12(0),D,B,C);
719	3	CALL LOAD\$DATA\$BIGPICTURE(.L13(0),D,B,C);
720	3	CALL LOAD\$DATA\$BIGPICTURE(.L14(0),D,B,C);
721	3	CALL LOAD\$DATA\$BIGPICTURE(.L15(0),D,B,C);
722	3	CALL LOAD\$DATA\$BIGPICTURE(.L16(0),D,B,C);
723	3	CALL LOAD\$DATA\$BIGPICTURE(.L17(0),D,B,C);
724	3	CALL LOAD\$DATA\$BIGPICTURE(.L18(0),D,B,C);


```

725 3      CALL LOAD$DATA$BIGPICTURE(.L15(0),D,B,C);
726 3      CALL LOAD$DATA$BIGPICTURE(.L16(0),D,B,C);
727 3      CALL LOAD$DATA$BIGPICTURE(.L17(0),D,B,C);
728 3      END; /* CASE A */
729 2      END; /* LOAD$LINE$ARRAY */

730 1      BLANK: PROCEDURE (PTR,P$COUNT);

731 2      /* THIS PROCEDURE LOADS BLANKS IN THE BIG PICTURE */
732 2      DCL (PTR,P$COUNT) ADDRESS ;
733 2      DCL I BYTE ;
734 2      DCL P BASED PTR BYTE ;
735 3      DO I=0 TO (P$COUNT-1);
(P='H') OR (P - = 'O') OR (P='U') OR (P='N') OR (P='P') OR (P='S')) THE
N
736 3      P=' ';
737 3      PTR = PTR + 1;
738 3      END; /* DO I */

```


739 2 END; /* BLANK */

740 1 LOAD\$BLANKS: PROCEDURE (BL , N) ;

/* THIS PROCEDURE LOCATES APPROPRIATE LINE ARRAY FOR B

LANKING */

```

741 2 DCL (BL,N) ADDRESS ;
742 2 DO CASE BL;
743 3 CALL BLANK(.L0(0),N);
744 3 CALL BLANK(.L1(0),N);
745 3 CALL BLANK(.L2(0),N);
746 3 CALL BLANK(.L3(0),N);
747 3 CALL BLANK(.L4(0),N);
748 3 CALL BLANK(.L5(0),N);
749 3 CALL BLANK(.L6(0),N);
750 3 CALL BLANK(.L7(0),N);
751 3 CALL BLANK(.L8(0),N);
752 3 CALL BLANK(.L9(0),N);
753 3 CALL BLANK(.L10(0),N);
754 3 CALL BLANK(.L11(0),N);

```



```

755      3      CALL BLANK(.L12(0),N);
756      3      CALL BLANK(.L13(0),N);
757      3      CALL BLANK(.L14(0),N);
758      3      CALL BLANK(.L15(0),N);
759      3      CALL BLANK(.L16(0),N);
760      3      CALL BLANK(.L17(0),N);
761      3      END; /* CASE */
762      2      END; /* LOAD$BLANKS */

```

```

763      1      INITIALIZE$LOAD$PICTURE: PROCEDURE (C) PUBLIC ;

          /* THIS PROCEDURE INITIALIZES TRANSFER OF STRUCTURE D
          TO CRT-LINE PRINTER STATISTICAL INFORMATION. */

          DCL (C,F) ADDRESS ;
          DCL (INDEX,SHIPINDEX,D,E) BYTE ;
          DCL T$COUNT ADDRESS ;
          T$COUNT=C;
          D=0;
          F=73;

```



```
770 SHIPINDEX=0 ; /* SHIPINDEX */
771 DO INDEX=0 TO 2 ;
    /* INDEX */
772 IF (SHIP$PLOT(INDEX).COUNT) > 0 THEN
773 DO WHILE SHIPINDEX < SHIP$PLOT(INDEX).COUNT;
774 CALL LOAD$LINE$ARRAY(T$COUNT,INDEX,SHIPINDEX);
775 D=D+1;
776 T$COUNT = T$COUNT + 1;
777 SHIPINDEX =SHIPINDEX + 1;
778 END ; /* DO WHILE J */
779 SHIPINDEX=0;
780 END; /* DO INDEX */
781 IF ERASE$COUNT <= D THEN
782 ERASE$COUNT = D;
783 ELSE DO;
784 DO E = 1 TO (ERASE$COUNT-D);
785 CALL LOAD$BLANKS(T$COUNT,F);
786 T$COUNT=T$COUNT+1;
787 END; /* DO E */
788 ERASE$COUNT = D;
789 END; /* ELSE */
```



```
790 2      END; /* INITIALIZE$LOAD$PICTURE */

791 1      SET$STRUCTURE$ZEREOES: PROCEDURE (PTR,P$COUNT) PUBLIC;
      2      /* THIS PROCEDURE SETS THE STRUCTURE TO ZEROES */
      2      DCL (PTR,P$COUNT) ADDRESS;
793 2      DCL ZERO$COUNTER ADDRESS;
794 2      DCL P BASED PTR BYTE;
795 2      DO ZERO$COUNTER=0 TO (P$COUNT - 1);
796 3      P=0;
797 3      PTR=PTR+1;
798 3      END; /* DO */
799 2      END; /* SET$STRUCTURE$ZEREOES */

800 1      END; /* SCREEN$MODULE */
```


MODULE INFORMATION:

CODE AREA SIZE	= 179EH	6046D
VARIABLE AREA SIZE	= 09E4H	2532D
MAXIMUM STACK SIZE	= 0008H	8D
1209 LINES READ		
0 PROGRAM ERROR(S)		

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE GRAPH1MODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:GRAPH1.MOD NOOBJECT PAGEWIDTH(82) PA
GELENGTH(24) D
--ATE(MAY 1979) TITLE('MODULE GRAPH1.MOD')

/* THIS MODULE CONTAINS PROCEDURES TO DRAW SYMBOLS TO
PLASMA DEVICES AND INTERACT WITH PLASMA TOUCH--PAN

THE

EL. */

```

1      GRAPH1$MODULE:
      DO;
2      1  DECLARE SHIP$PLOT (15) STRUCTURE (
          LAT(10) ADDRESS,
          LONG(10) ADDRESS,
          COURSE (10) ADDRESS,
          SPEED(10) BYTE,
          X$BOW(10) ADDRESS,
          Y$BOW(10) ADDRESS,
          QUADRANT(10) BYTE,
          RANGE(10) ADDRESS,

```



```

3 1      BEARING(10) ADDRESS,
4 1      COLLISION$FLAG(10) BYTE,
5 1      CPA$TIME(10) ADDRESS,
6 1      CPA$DISTANCE(10) ADDRESS,
7 1      COUNT BYTE ) PUBLIC ;
8 1      DECLARE JSPECIAL ADDRESS;
9 1      DECLARE BACKUP$SHIP$COUNT BYTE;
10 1      DECLARE LIT LITERALLY 'LITERALLY';
11 1      DECLARE DCL LIT 'DECLARE';
12 1      DCL TRUE LIT 'OFFH';
13 1      DCL FALSE LIT '000H';

```

```

$NOLIST
$INCLUDE(:F1:PLAPUB.EXT)

```

= = = = =

/* EXTERNAL PLASMA PUBLICS FOR PLASMA.DEVICE-ONE . */


```

237 1 = WRITE$CONTACT$ID: PROCEDURE (X,Y,J) EXTERNAL ;
238 2 = DCL (ROW,COLUMN,J) BYTE ;
239 2 = DCL (X,Y) ADDRESS ;
240 2 = DCL CONTACT$ID (3) BYTE ;
241 2 = END;
=
=
242 1 = DRAW$GRID: PROCEDURE EXTERNAL;
243 2 = DCL (X,Y) ADDRESS;
244 2 = END;
=
=
245 1 = DRAW$FRIEND$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
246 2 = DCL (X,Y,TX,TY) ADDRESS;
247 2 = DCL S BYTE;
248 2 = END;
=
=
249 1 = DRAW$FRIEND$DASH: PROCEDURE (X,Y,S) EXTERNAL;
250 2 = DCL (X,Y,TX,TY) ADDRESS;

```


MAY 1

```

251      =      DCL S BYTE;
252      =      END;
      =
      =
253      1      ERASE$FRIEND$DASH: PROCEDURE (X,Y,S) EXTERNAL;
254      2      DCL (X,Y,TX,TY) ADDRESS;
255      2      DCL S BYTE;
256      2      END;
      =
      =
257      1      ERASE$FRIEND$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
258      2      DCL (X,Y,TX,TY) ADDRESS;
259      2      DCL S BYTE;
260      2      END;
      =
      =
261      1      DRAW$HOSTILE$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
262      2      DCL (X,Y,TX,TY) ADDRESS;
263      2      DCL S BYTE;
264      2      END;
      =

```



```

=
265 1 = DRAW$HOSTILE$DASH: PROCEDURE (X,Y,S) EXTERNAL;
266 2 = DCL (X,Y,TX,TY) ADDRESS;
267 2 = DCL S BYTE;
268 2 = END;
=
=
269 1 = ERASE$HOSTILE$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
270 2 = DCL (X,Y,TX,TY) ADDRESS;
271 2 = DCL S BYTE;
272 2 = END;
=
=
273 1 = ERASE$HOSTILE$DASH: PROCEDURE (X,Y,S) EXTERNAL;
274 2 = DCL (X,Y,TX,TY) ADDRESS;
275 2 = DCL S BYTE;
276 2 = END;
=
=
277 1 = DRAW$UNKNOWN$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
278 2 = DCL (X,Y,TX,TY) ADDRESS;

```


279	2	=	DCL S BYTE;
280	2	=	END;
		=	
281	1	=	ERASE\$UNKNOWN\$SYMBOL: PROCEDURE (X,Y,S) EXTERNAL;
282	2	=	DCL (X,Y,TX,TY) ADDRESS;
283	2	=	DCL S BYTE;
284	2	=	END;
		=	
285	1	=	DRAW\$UNKNOWN\$DASH: PROCEDURE (X,Y,S) EXTERNAL;
286	2	=	DCL (X,Y,TX,TY) ADDRESS;
287	2	=	DCL S BYTE;
288	2	=	END;
		=	
289	1	=	ERASE\$UNKNOWN\$DASH: PROCEDURE (X,Y,S) EXTERNAL;
290	2	=	DCL (X,Y,TX,TY) ADDRESS;
291	2	=	DCL S BYTE;
292	2	=	END;
		=	


```

=
=
=
=
=
293      1      WRITE$CONTACT$ID$2: PROCEDURE (X,Y,J) EXTERNAL ;
294      2      DCL (ROW,COLUMN,J) BYTE ;
295      2      DCL (X,Y) ADDRESS ;
296      2      DCL CONTACT$ID (3) BYTE ;
297      2      END;
=
=
=
=
=
298      1      DRAW$GRID$2: PROCEDURE EXTERNAL;
299      2      DCL (X,Y) ADDRESS;
300      2      END;
=
=
=
=
=
301      1      DRAW$FRIEND$SYMBOL$2: PROCEDURE (X,Y,S) EXTERNAL;
302      2      DCL (X,Y,TX,TY) ADDRESS;
303      2      DCL S BYTE;

```


MAY 1

```

304 2 = END;
    =
    =
305 1 = DRAW$FRIEND$DASH$2: PROCEDURE (X,Y,S) EXTERNAL;
306 2 =   DCL (X,Y,TX,TY) ADDRESS;
307 2 =   DCL S BYTE;
308 2 = END;
    =
    =
309 1 = ERASE$FRIEND$DASH$2: PROCEDURE (X,Y,S) EXTERNAL;
310 2 =   DCL (X,Y,TX,TY) ADDRESS;
311 2 =   DCL S BYTE;
312 2 = END;
    =
    =
313 1 = ERASE$FRIEND$SYMBOL$2: PROCEDURE (X,Y,S) EXTERNAL;
314 2 =   DCL (X,Y,TX,TY) ADDRESS;
315 2 =   DCL S BYTE;
316 2 = END;
    =
    =

```



```

317 1 = DRAW$HOSTILE$SYMBOL$2: PROCEDURE (X,Y,S) EXTERNAL;
318 2 = DCL (X,Y,TX,TY) ADDRESS;
319 2 = DCL S BYTE;
320 2 = END;
    =
    =
321 1 = DRAW$HOSTILE$DASH$2: PROCEDURE (X,Y,S) EXTERNAL;
322 2 = DCL (X,Y,TX,TY) ADDRESS;
323 2 = DCL S BYTE;
324 2 = END;
    =
    =
325 1 = ERASE$HOSTILE$SYMBOL$2: PROCEDURE (X,Y,S) EXTERNAL;
326 2 = DCL (X,Y,TX,TY) ADDRESS;
327 2 = DCL S BYTE;
328 2 = END;
    =
    =
329 1 = ERASE$HOSTILE$DASH$2: PROCEDURE (X,Y,S) EXTERNAL;
330 2 = DCL (X,Y,TX,TY) ADDRESS;
331 2 = DCL S BYTE;

```


332	2	=	END;
		=	
		=	
333	1	=	DRAW\$UNKNOWN\$SYMBOL\$2: PROCEDURE (X,Y,S) EXTERNAL;
334	2	=	DCL (X,Y,TX,TY) ADDRESS;
335	2	=	DCL S BYTE;
336	2	=	END;
		=	
		=	
337	1	=	ERASE\$UNKNOWN\$SYMBOL\$2: PROCEDURE (X,Y,S) EXTERNAL;
338	2	=	DCL (X,Y,TX,TY) ADDRESS;
339	2	=	DCL S BYTE;
340	2	=	END;
		=	
		=	
341	1	=	DRAW\$UNKNOWN\$DASH\$2: PROCEDURE (X,Y,S) EXTERNAL;
342	2	=	DCL (X,Y,TX,TY) ADDRESS;
343	2	=	DCL S BYTE;
344	2	=	END;
		=	
		=	


```
345 1 = FRASE$UNKNOWN$DASH$2: PROCEDURE (X,Y,S) EXTERNAL;  
346 2 = DCL (X,Y,TX,TY) ADDRESS;  
347 2 = DCL S BYTE;  
348 2 = END;  
= =  
= =  
349 1 = $INCLUDE(:F1:SERVE.SIX)  
= = SERVICE$SIX: PROCEDURE PUBLIC;  
= =  
= = /* THIS PROCEDURE SERVICE THE INTERRUPTS CAUSED BY  
= = THE PLASMA(DEVICE ONE) TOUCH-PANEL DEVICE. */  
= =  
350 2 = DCL (XY$DATAWORD,XY$DATAWORD$1,X$COORD,Y$COORD,INDEX,  
351 2 = SHIPINDEX,FOUND$SHIP,QUADRANT$NUM,OWNSHIP) BYTE;  
352 2 = OWNSHIP = 1; /* INITIALIZE DESIGNATOR OF OWN SHIP */  
353 2 = XY$DATAWORD=INPUT(7);  
354 2 = XY$DATAWORD$1=XY$DATAWORD;  
355 2 = X$COORD=XY$DATAWORD AND 0FH;  
356 2 = Y$COORD=FOR(NOT(XY$DATAWORD),4)AND 0FH;  
357 2 = CALL CLEAR$PLASMA$2;  
= IF (X$COORD=>=0) AND (X$COORD<=3) THEN
```



```

358      = DO;
359      3 = IF (Y$COOR>=0) AND (Y$COOR<=3) THEN
360      3 =   QUADRANT$NUM=1;
           ELSE
361      3 = IF (Y$COOR>=4) AND (Y$COOR<=7) THEN
362      3 =   QUADRANT$NUM=5;
           ELSE
363      3 = IF (Y$COOR>=8) AND (Y$COOR<=11) THEN
364      3 =   QUADRANT$NUM=9;
           ELSE
365      3 = IF (Y$COOR>=12) AND (Y$COOR<=15) THEN
366      3 =   QUADRANT$NUM=13;
           END;
           ELSE
368      2 = IF (X$COOR>=4) AND (X$COOR<=7) THEN
369      2 = DO;
370      3 = IF (Y$COOR>=0) AND (Y$COOR<=3) THEN
371      3 =   QUADRANT$NUM=2;
           ELSE
372      3 = IF (Y$COOR>=4) AND (Y$COOR<=7) THEN
373      3 =   QUADRANT$NUM=6;

```



```

374      =      ELSE
375      3      IF (Y$COOR>=8) AND (Y$COOR<=11) THEN
376      3      QUADRANT$NUM=10;
377      3      ELSE
378      3      IF (Y$COOR>=12) AND (Y$COOR<=15) THEN
379      3      QUADRANT$NUM=14;
380      3      END;
381      3      ELSE
382      3      IF (X$COOR>=8) AND (X$COOR<=11) THEN
383      3      DO;
384      3      IF (Y$COOR>=0) AND (Y$COOR<=3) THEN
385      3      QUADRANT$NUM=3;
386      3      ELSE
387      3      IF (Y$COOR>=4) AND (Y$COOR<=7) THEN
388      3      QUADRANT$NUM=7;
389      3      ELSE
390      3      IF (Y$COOR>=8) AND (Y$COOR<=11) THEN
391      3      QUADRANT$NUM=11;
392      3      ELSE
393      3      IF (Y$COOR>=12) AND (Y$COOR<=15) THEN

```



```

388 3 = QUADRANT$NUM=15;
      = END;
      = ELSE
      =
390 2 = IF (X$COOR>=12) AND (X$COOR<=15) THEN
391 2 = DO;
392 3 = IF (Y$COOR>=0) AND (Y$COOR<=3) THEN
393 3 = QUADRANT$NUM=4;
      = ELSE
394 3 = IF (Y$COOR>=4) AND (Y$COOR<=7) THEN
395 3 = QUADRANT$NUM=8;
      = ELSE
396 3 = IF (Y$COOR>=8) AND (Y$COOR<=11) THEN
397 3 = QUADRANT$NUM=12;
      = ELSE
398 3 = IF (Y$COOR>=12) AND (Y$COOR<=15) THEN
399 3 = QUADRANT$NUM=16;
      = END;
      = INDEX=0;
      = SHIPINDEX=0;
402 2 = DO WHILE INDEX < 3;
403 2 =

```



```

404 3 =
405 3 =
406 4 =
$NUM) THEN
407 4 =
408 5 =
409 5 =
410 5 =
      IF(SHIP$PLOT(INDEX).COUNT) > 0 THEN
        DO WHILE SHIPINDEX < SHIP$PLOT(INDEX).COUNT;
          IF(SHIP$PLOT(INDEX).QUADRANT(SHIPINDEX)=QUADRANT
DO;
      FOUND$SHIP=TRUE;
      IF INDEX=0 THEN
        CALL DRAW$FRIEND$SYMBOL$2(
          SHIP$PLOT(0).X$BOW(SHIPINDEX),
          SHIP$PLOT(0).Y$BOW(SHIPINDEX),1);
      ELSE
        IF INDEX=1 THEN
          CALL DRAW$HOSTILE$SYMBOL$2(
            SHIP$PLOT(1).X$BOW(SHIPINDEX),
            SHIP$PLOT(1).Y$BOW(SHIPINDEX),1);
          ELSE
            IF INDEX=2 THEN
              CALL DRAW$UNKNOWN$SYMBOL$2(
                SHIP$PLOT(2).X$BOW(SHIPINDEX),
                SHIP$PLOT(2).Y$BOW(SHIPINDEX),1);
/* WRITE$CONTACT$ID NEXT TO SYMBOL */

```



```

= =
THEN
416 5 =
X$BOW(SHIPINDE
-
X)+11,
=
SHIPINDEX);
=
417 5 =
X$BOW(SHIPINDE
-
X)-18,
=
SHIPINDEX);
=
418 5 =
=
=
=
419 5 =
=
=
=
420 5 =
421 4 =
422 4 =
424 4 =
IF (SHIP$PLOT(INDEX).X$BOW(SHIPINDEX) <= 496)
CALL WRITE$CONTACT$ID$2(SHIP$PLOT(INDEX)).
SHIP$PLOT(INDEX).Y$BOW(SHIPINDEX)+8.
ELSE
CALL WRITE$CONTACT$ID$2(SHIP$PLOT(INDEX)).
SHIP$PLOT(INDEX).Y$BOW(SHIPINDEX)+8.
CALL START$VECTOR$DASH$2(
SHIP$PLOT(0).X$BOW(OWNSHIP),
SHIP$PLOT(0).Y$BOW(OWNSHIP));
CALL STOP$VECTOR$DASH$2(
SHIP$PLOT(INDEX).X$BOW(SHIPINDEX),
SHIP$PLOT(INDEX).Y$BOW(SHIPINDEX));
END;
SHIPINDEX=SHIPINDEX+1;
IF (INDEX=0) AND (SHIPINDEX = OWNSHIP)
THEN SHIPINDEX = SHIPINDEX + 1;
END; /* DO WHILE SHIPINDEX */

```



```

=
=
436 1 = QUAD$NUM: PROCEDURE (X,Y) BYTE PUBLIC;
=
= /* THIS PROCEDURE CONVERTS ABSOLUTE X,Y COORDINATES O
=
= THE PLASMA (DEVICE ONE) TO QUADRANT NUMBERS AS DEF
=
= NED FOR THE TOUCH-PANEL . */
=
= DECLARE (X,Y) ADDRESS ;
437 2 = DECLARE QUADRANT$NUM BYTE ;
438 2 =
439 2 = IF (X>=0) AND (X<=127) THEN
440 2 = DO;
441 3 = IF (Y>=0) AND (Y<=127) THEN
442 3 = QUADRANT$NUM=1;
=
= ELSE
443 3 = IF (Y>=128) AND (Y<=255) THEN
444 3 = QUADRANT$NUM=5;
=
= ELSE
445 3 = IF (Y>=256) AND (Y<=383) THEN
446 3 = QUADRANT$NUM=9;
=
= ELSE
447 3 = IF (Y>=384) AND (Y<=511) THEN

```



```

448      3      =      QUADRANT$NUM=13;
      3      =      END;
      3      =      ELSE
450      2      =      IF (X>=128) AND (X<=255) THEN
451      2      =      DO;
452      3      =      IF (Y>=0) AND (Y<=127) THEN
453      3      =      QUADRANT$NUM=2;
      3      =      ELSE
454      3      =      IF (Y>=128) AND (Y<=255) THEN
455      3      =      QUADRANT$NUM=6;
      3      =      ELSE
456      3      =      IF (Y>=256) AND (Y<=383) THEN
457      3      =      QUADRANT$NUM=10;
      3      =      ELSE
458      3      =      IF (Y>=384) AND (Y<=511) THEN
459      3      =      QUADRANT$NUM=14;
      3      =      END;
      3      =      ELSE
      3      =
461      2      =      IF (X>=256) AND (X<=383) THEN
462      2      =      DO;

```



```

463      = IF (Y>=0) AND (Y<=127) THEN
464      3   QUADRANT$NUM=3;
465      = ELSE
466      3   IF (Y>=128) AND (Y<=255) THEN
467      3   QUADRANT$NUM=7;
468      = ELSE
469      3   IF (Y>=256) AND (Y<=383) THEN
470      3   QUADRANT$NUM=11;
471      = ELSE
472      3   IF (Y>=384) AND (Y<=511) THEN
473      3   QUADRANT$NUM=15;
474      = END;
475      = ELSE
476      =
477      =
478      =
479      =
480      =
481      =
482      =
483      =
484      =
485      =
486      =
487      =
488      =
489      =
490      =
491      =
492      =
493      =
494      =
495      =
496      =
497      =
498      =
499      =
500      =
501      =
502      =
503      =
504      =
505      =
506      =
507      =
508      =
509      =
510      =
511      =
512      =
513      =
514      =
515      =
516      =
517      =
518      =
519      =
520      =
521      =
522      =
523      =
524      =
525      =
526      =
527      =
528      =
529      =
530      =
531      =
532      =
533      =
534      =
535      =
536      =
537      =
538      =
539      =
540      =
541      =
542      =
543      =
544      =
545      =
546      =
547      =
548      =
549      =
550      =
551      =
552      =
553      =
554      =
555      =
556      =
557      =
558      =
559      =
560      =
561      =
562      =
563      =
564      =
565      =
566      =
567      =
568      =
569      =
570      =
571      =
572      =
573      =
574      =
575      =
576      =
577      =
578      =
579      =
580      =
581      =
582      =
583      =
584      =
585      =
586      =
587      =
588      =
589      =
590      =
591      =
592      =
593      =
594      =
595      =
596      =
597      =
598      =
599      =
600      =
601      =
602      =
603      =
604      =
605      =
606      =
607      =
608      =
609      =
610      =
611      =
612      =
613      =
614      =
615      =
616      =
617      =
618      =
619      =
620      =
621      =
622      =
623      =
624      =
625      =
626      =
627      =
628      =
629      =
630      =
631      =
632      =
633      =
634      =
635      =
636      =
637      =
638      =
639      =
640      =
641      =
642      =
643      =
644      =
645      =
646      =
647      =
648      =
649      =
650      =
651      =
652      =
653      =
654      =
655      =
656      =
657      =
658      =
659      =
660      =
661      =
662      =
663      =
664      =
665      =
666      =
667      =
668      =
669      =
670      =
671      =
672      =
673      =
674      =
675      =
676      =
677      =
678      =
679      =
680      =
681      =
682      =
683      =
684      =
685      =
686      =
687      =
688      =
689      =
690      =
691      =
692      =
693      =
694      =
695      =
696      =
697      =
698      =
699      =
700      =
701      =
702      =
703      =
704      =
705      =
706      =
707      =
708      =
709      =
710      =
711      =
712      =
713      =
714      =
715      =
716      =
717      =
718      =
719      =
720      =
721      =
722      =
723      =
724      =
725      =
726      =
727      =
728      =
729      =
730      =
731      =
732      =
733      =
734      =
735      =
736      =
737      =
738      =
739      =
740      =
741      =
742      =
743      =
744      =
745      =
746      =
747      =
748      =
749      =
750      =
751      =
752      =
753      =
754      =
755      =
756      =
757      =
758      =
759      =
760      =
761      =
762      =
763      =
764      =
765      =
766      =
767      =
768      =
769      =
770      =
771      =
772      =
773      =
774      =
775      =
776      =
777      =
778      =
779      =
780      =
781      =
782      =
783      =
784      =
785      =
786      =
787      =
788      =
789      =
790      =
791      =
792      =
793      =
794      =
795      =
796      =
797      =
798      =
799      =
800      =
801      =
802      =
803      =
804      =
805      =
806      =
807      =
808      =
809      =
810      =
811      =
812      =
813      =
814      =
815      =
816      =
817      =
818      =
819      =
820      =
821      =
822      =
823      =
824      =
825      =
826      =
827      =
828      =
829      =
830      =
831      =
832      =
833      =
834      =
835      =
836      =
837      =
838      =
839      =
840      =
841      =
842      =
843      =
844      =
845      =
846      =
847      =
848      =
849      =
850      =
851      =
852      =
853      =
854      =
855      =
856      =
857      =
858      =
859      =
860      =
861      =
862      =
863      =
864      =
865      =
866      =
867      =
868      =
869      =
870      =
871      =
872      =
873      =
874      =
875      =
876      =
877      =
878      =
879      =
880      =
881      =
882      =
883      =
884      =
885      =
886      =
887      =
888      =
889      =
890      =
891      =
892      =
893      =
894      =
895      =
896      =
897      =
898      =
899      =
900      =
901      =
902      =
903      =
904      =
905      =
906      =
907      =
908      =
909      =
910      =
911      =
912      =
913      =
914      =
915      =
916      =
917      =
918      =
919      =
920      =
921      =
922      =
923      =
924      =
925      =
926      =
927      =
928      =
929      =
930      =
931      =
932      =
933      =
934      =
935      =
936      =
937      =
938      =
939      =
940      =
941      =
942      =
943      =
944      =
945      =
946      =
947      =
948      =
949      =
950      =
951      =
952      =
953      =
954      =
955      =
956      =
957      =
958      =
959      =
960      =
961      =
962      =
963      =
964      =
965      =
966      =
967      =
968      =
969      =
970      =
971      =
972      =
973      =
974      =
975      =
976      =
977      =
978      =
979      =
980      =
981      =
982      =
983      =
984      =
985      =
986      =
987      =
988      =
989      =
990      =
991      =
992      =
993      =
994      =
995      =
996      =
997      =
998      =
999      =
1000     =

```


490 2 CALL INITIALIZE\$PLASMA\$2;
 491 2 CALL CLEAR\$PLASMA\$2;
 492 2 END; /* INITIALIZE\$GRAPHICS */

493 1 SEND\$GRID\$PLASMA: PROCEDURE PUBLIC;
 494 2 CALL CLEAR\$PLASMA;
 495 2 CALL DRAW\$GRID;
 496 2 END;

497 1 SEND\$GRID\$PLASMA\$2: PROCEDURE PUBLIC;
 498 2 CALL CLEAR\$PLASMA\$2;
 499 2 CALL DRAW\$GRID\$2;
 500 2 END;

501 1 INITIALIZE\$STRUCTURE: PROCEDURE PUBLIC;

/* THIS A PROCEDURE TO INITIALIZE STRUCTURE TO TEST MO
 DULE STAND-ALO
 - NE. */


```

502      DCL (A0,B0,C0,D0,E0,F0,G0,H0) ADDRESS;
503      A0=100;
504      B0=200;
505      C0=300;
506      D0=400;
507      G0=275;
508      H0=350;
509      IF (JSPECIAL =100) THEN
510      DO;
511      F0=0;
512      E0=0;
513      END;
514      SHIP$PLOT(1).COUNT=2;
515      SHIP$PLOT(0).COUNT=3;
516      SHIP$PLOT(2).COUNT=1;
517      SHIP$PLOT(0).X$BOW(0)=A0;
518      SHIP$PLOT(0).X$BOW(1)=D0;
519      SHIP$PLOT(0).X$BOW(2)=G0;
520      SHIP$PLOT(1).X$BOW(0)=B0;
521      SHIP$PLOT(1).X$BOW(1)=H0;
522      SHIP$PLOT(2).X$BOW(0)=C0;

```



```

523 2 SHIP$PLOT(0).Y$BOW(0)=479-(1*E0);
524 2 SHIP$PLOT(0).Y$BOW(1)=351-(2*E0);
525 2 SHIP$PLOT(0).Y$BOW(2)=447-(2*E0);
526 2 SHIP$PLOT(1).Y$BOW(0)=479-(3*E0);
527 2 SHIP$PLOT(1).Y$BOW(1)=447-(2*E0);
528 2 SHIP$PLOT(2).Y$BOW(0)=479-(4*E0);
529 2 SHIP$PLOT(0).QUADRANT(0)=QUAD$NUM(SHIP$PLOT(0)).X$BOW(0
),SHIP$PLOT(0)
- .Y$BOW(0));
530 2 SHIP$PLOT(0).QUADRANT(1)=QUAD$NUM(SHIP$PLOT(0)).X$BOW(1
),SHIP$PLOT(0)
- .Y$BOW(1));
531 2 SHIP$PLOT(0).QUADRANT(2)=QUAD$NUM(SHIP$PLOT(0)).X$BOW(2
),SHIP$PLOT(0)
- .Y$BOW(2));
532 2 SHIP$PLOT(1).QUADRANT(0)=QUAD$NUM(SHIP$PLOT(1)).X$BOW(0
),SHIP$PLOT(1)
- .Y$BOW(0));
533 2 SHIP$PLOT(1).QUADRANT(1)=QUAD$NUM(SHIP$PLOT(1)).X$BOW(1
),SHIP$PLOT(1)
- .Y$BOW(1));
534 2 SHIP$PLOT(2).QUADRANT(0)=QUAD$NUM(SHIP$PLOT(2)).X$BOW(0
),SHIP$PLOT(2)
- .Y$BOW(0));
535 2 E0=E0+1;
536 2 JSPECIAL=JSPECIAL+1;
537 2 END; /* INITIALIZE$STRUCTURE */

```



```
538 1 DRAW$SHIP$DASH: PROCEDURE (A1) PUBLIC;  
    DISPLAY  
    /* THIS PROCEDURE DRAWS DASHED SHIPS IN THE PLASMA  
       (DEVICE ONE). */  
539 2 DCL (A1,SHIPINDEX,INDEX) BYTE ;  
540 2 INDEX=0;  
541 2 SHIPINDEX=0;  
542 2 DO WHILE INDEX < 3 ;  
543 3 IF (SHIP$PLOT(INDEX + A1).COUNT) > 0 THEN  
544 3 DO WHILE SHIPINDEX < SHIP$PLOT(INDEX + A1).COUNT  
    ;  
545 4 IF INDEX = 0 THEN  
546 4 CALL DRAW$FRIEND$DASH(  
    SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),  
    SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);  
    ELSE  
547 4 IF INDEX = 1 THEN  
548 4 CALL DRAW$HOSTILE$DASH(  
    SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),  
    SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);
```



```

549 4 ELSE
550 4 IF INDEX = 2 THEN
      CALL DRAW$UNKNOWN$DASH(
        SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),
        SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);
        SHIPINDEX=SHIPINDEX + 1;
        /* AD ANNOTATION ROUTINE *****/
552 4 END; /* DO WHILE SHIPINDEX **/
553 3 SHIPINDEX = 0;
554 3 INDEX = INDEX + 1;
555 3 END; /* DO WHILE INDEX < 3 */
556 2 END; /* DRAW$SHIP$DASH */

557 1 DRAW$SHIP$DASH$LOOP: PROCEDURE (A2) PUBLIC;

      /* THIS PROCEDURE DRAWS BACKUP DASH SHIP POSITIONS
*/
558 2 DCL (A2,B2,C2) BYTE ;
559 2 C2 = 0 ;
560 2 DO B2 = 1 TO A2 ;
561 3 CALL DRAW$SHIP$DASH (3 + C2);
```



```

562      3      C2=C2 + 3;
563      3      END; /* DO B2 */
564      2      END; /* DRAW$SHIP$DASH$LOOP */

565      1      STRUCTURE$BACK: PROCEDURE (A3) PUBLIC;

                /* THIS PROCEDURE TRANSFERS SHIP STRUCTURE SET (3) T
                ADJACENT BACKUP SHIP STRUCTURE SET. */
566      2      DCL (A3,B,C,D,E) BYTE;
567      2      B=A3 + 2;
568      2      E=A3 + 3;
569      2      DO C = A3 TO B;
570      3      SHIP$PLOT(E).COUNT = SHIP$PLOT(C).COUNT;
571      3      IF SHIP$PLOT(E).COUNT > 0 THEN
572      3      DO D = 0 TO (SHIP$PLOT(E).COUNT-1);
573      4      SHIP$PLOT(E).LAT(D)=SHIP$PLOT(C).LAT(D);
574      4      SHIP$PLOT(E).LONG(D)=SHIP$PLOT(C).LONG(D);
575      4      SHIP$PLOT(E).COURSE(D)=SHIP$PLOT(C).COURSE(D);
576      4      SHIP$PLOT(E).SPEED(D)=SHIP$PLOT(C).SPEED(D);
577      4      SHIP$PLOT(E).X$BOW(D)=SHIP$PLOT(C).X$BOW(D);

```

0 NEXT


```

578 4 SHIP$PLOT(E).Y$BOW(D)=SHIP$PLOT(C).Y$BOW(D);
579 4 SHIP$PLOT(E).QUADRANT(D)=SHIP$PLOT(C).QUADRANT(D)
;
580 4 SHIP$PLOT(E).RANGE(D)=SHIP$PLOT(C).RANGE(D);
581 4 SHIP$PLOT(E).BEARING(D)=SHIP$PLOT(C).BEARING(D);
582 4 SHIP$PLOT(E).COLLISION$FLAG(D)=SHIP$PLOT(C).COLLI
SION$FLAG(D);
583 4 SHIP$PLOT(E).CPA$TIME(D)=SHIP$PLOT(C).CPA$TIME(D)
;
584 4 SHIP$PLOT(E).CPA$DISTANCE(D)=SHIP$PLOT(C).CPA$DIS
TANCE(D);
585 4
/* DO D=1 TO SHIP$PLOT */
586 3 E=E + 1;
587 3 END; /* DO C = A3 TO B */
588 2 END; /* STRUCTURE$BACK */

589 1 BACKUPS: PROCEDURE (A4) PUBLIC;

/* THIS PROCEDURE TRANSFERS BACKUP STRUCTURES IN FIFO
QUEUE. */
590 2 DCL (A4,B4,C4) BYTE;
591 2 BACKUP$SHIP$COUNT=BACKUP$SHIP$COUNT + 1;
592 2 IF BACKUP$SHIP$COUNT = 5 THEN
593 2 DO ;

```



```

*****/
594 3      BACKUP$SHIP$COUNT=0;
595 3      END; /* END IF */
596 2      DO CASE A4-1;
597 3          B4=0;
598 3          B4=3;
599 3          B4=6;
600 3          B4=9;
601 3      END; /* CASE A4 */
602 2      DO C4 = 1 TO A4;
603 3          CALL STRUCTURE$BACK(B4);
604 3          B4 = B4 - 3;
605 3      END; /* DO C4 = 1 TO A4 */
606 2      END; /* BACKUPS */

607 1      DRAW$SHIP:PROCEDURE (TAIL) PUBLIC;

/*THIS PROCEDURE DRAW SHIPS IN THE PLASMA DISPLAY (DEV
ICE ONE) */
608 2      DCL (A,TAIL) BYTE;
609 2      DCL (X1,Y1,X2,Y2,B,C) ADDRESS;

```



```
610 2 DCL (ZINDEX,ZSHIPINDEX) BYTE;  
611 2 ZINDEX=0;  
612 2 ZSHIPINDEX=0;  
613 2 DO WHILE ZINDEX < 3 ;  
614 3 IF (SHIP$PLOT(ZINDEX).COUNT) > 0 THEN  
615 3 DO WHILE ZSHIPINDEX < SHIP$PLOT(ZINDEX).COUNT;  
616 4 IF ZINDEX=0 THEN  
617 4 CALL DRAW$FRIEND$SYMBOL(  
    SHIP$PLOT(0).X$BOW(ZSHIPINDEX),  
    SHIP$PLOT(0).Y$BOW(ZSHIPINDEX),1);  
    ELSE  
    IF ZINDEX=1 THEN  
    CALL DRAW$HOSTILE$SYMBOL(  
    SHIP$PLOT(1).X$BOW(ZSHIPINDEX),  
    SHIP$PLOT(1).Y$BOW(ZSHIPINDEX),1);  
    ELSE  
    IF ZINDEX=2 THEN  
    CALL DRAW$UNKNOWN$SYMBOL(  
    SHIP$PLOT(2).X$BOW(ZSHIPINDEX),  
    SHIP$PLOT(2).Y$BOW(ZSHIPINDEX),1);
```



```

623      4      /* DRAW TAIL FOR SHIP */
624      5      IF TAIL THEN
625      5      DO;
626      5      X1 = SHIP$PLOT(ZINDEX).X$BOW(ZSHIPINDEX);
        Y1 = SHIP$PLOT(ZINDEX).Y$BOW(ZSHIPINDEX);
        IF NOT ((X1<7) OR (X1>504) OR (Y1<7) OR (Y1>504)
        ) THEN
627      5      IF SHIP$PLOT(ZINDEX+3).COUNT > 0 THEN
628      5      DO;
629      6      IF SHIP$PLOT(ZINDEX+6).COUNT <= 0 THEN A=3;
631      6      ELSE IF SHIP$PLOT(ZINDEX+9).COUNT <= 0 THEN A
        =6;
633      6      ELSE IF SHIP$PLOT(ZINDEX+12).COUNT <= 0
        THEN A=9;
635      6      ELSE A=12;
636      6      X2 = SHIP$PLOT(ZINDEX+A).X$BOW(ZSHIPINDEX);
637      6      Y2 = SHIP$PLOT(ZINDEX+A).Y$BOW(ZSHIPINDEX);
638      6      B = X1 + (X2-X1) + (X2-X1) + (X2-X1);
639      6      C = Y1 + (Y2-Y1) + (Y2-Y1) + (Y2-Y1);
640      6      CALL STAPT$VECTOR$SOLID(X1,Y1);
641      6      CALL STOP$VECTOR$SOLID(B,C);
642      6      END; /* IF SHIP$PLOT(ZINDEX+3 */

```



```
643      5      END; /*IF TAIL */

644      4      X1=SHIP$PLOT(ZINDEX).X$BOW(ZSHIPINDEX);
645      4      Y1=SHIP$PLOT(ZINDEX).Y$BOW(ZSHIPINDEX);
646      4      IF((X1>=7) OR (X1<=504) OR (Y1>=7) OR (Y1<=504)) THEN
647      4      DO;
        /* WRITE CONTACT ID NEXT TO SYMBOL */
648      5      IF (SHIP$PLOT(ZINDEX).X$BOW(ZSHIPINDEX) <= 486)
        THEN
649      5      CALL WRITE$CONTACT$ID(SHIP$PLOT(ZINDEX).X$BOW
        (ZSHIPINDEX)+1
        -      1,
        HIPINDEX);
        -
        ELSE
        650      5      CALL WRITE$CONTACT$ID(SHIP$PLOT(ZINDEX).X$BOW
        (ZSHIPINDEX)-1
        -      8,
        HIPINDEX);
```



```

651      5      END;
652      4           ZSHIPINDEX=ZSHIPINDEX+1;
653      4           END; /* DO WHILE ZSHIPINDEX */
654      3           ZSHIPINDEX=0;
655      3           ZINDEX=ZINDEX+1;
656      3           END; /* DO WHILE ZINDEX < 3 */
657      2           END; /* DRAW$SHIP */

```

```

$INCLUDE(:F1:GRAPH2.SRC)

```

```

/* THIS MODULE CONTAINS PROCEDURES TO DRAW SYMBOLS TO

```

```

PLASMA DEVICE TWO */

```

```

/* GRAPH2$MODULE: */
DRAW$SHIP$DASH$2: PROCEDURE (A1) PUBLIC;

```

```

/* THIS PROCEDURE DRAWS DASHED SHIPS IN THE PLASMA
(DEVICE TWO). */
DCL (A1,SHIPINDEX,INDEX) BYTE ;

```



```

660 2 = INDEX=0;
661 2 = SHIPINDEX=0;
662 2 = DO WHILE INDEX < 3 ;
663 3 = IF (SHIP$PLOT(INDEX + A1).COUNT) > 0 THEN
664 3 = DO WHILE SHIPINDEX < SHIP$PLOT(INDEX + A1).COUNT
;
665 4 = IF INDEX = 0 THEN
666 4 = CALL DRAW$FRIEND$DASH$2(
SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),
SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);
ELSE
667 4 = IF INDEX = 1 THEN
668 4 = CALL DRAW$HOSTILE$DASH$2(
SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),
SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);
ELSE
669 4 = IF INDEX = 2 THEN
670 4 = CALL DRAW$UNKNOWN$DASH$2(
SHIP$PLOT(INDEX + A1).X$BOW(SHIPINDEX),
SHIP$PLOT(INDEX + A1).Y$BOW(SHIPINDEX),1);
SHIPINDEX=SHIPINDEX + 1;
672 4 = END; /* DO WHILE SHIPINDEX **/

```


MAY 1

```

673 3 = SHIPINDEX = 0;
674 3 = INDEX = INDEX + 1;
675 3 = END; /* DO WHILE INDEX < 3 */
676 2 = END; /* DRAW$SHIP$DASH$2 */
= =
= =
677 1 = DRAW$SHIP$DASH$LOOP$2: PROCEDURE (A2) PUBLIC;
= =
= = /* THIS PROCEDURE DRAWS BACKUP DASH SHIP POSITIONS
*/
678 2 = DCL (A2,B2,C2) BYTE ;
679 2 = C2 = 0 ;
680 2 = DO B2 = 1 TO A2 ;
681 3 = CALL DRAW$SHIP$DASH$2 (3 + C2);
682 3 = C2=C2 + 3;
683 3 = END; /* DO B2 */
684 2 = END; /* DRAW$SHIP$DASH$LOOP$2 */
= =
= =
685 1 = DRAW$SHIP$2:PROCEDURE (TAIL) PUBLIC;
= =
= = /*THIS PROCEDURE DRAW SHIPS IN THE PLASMA DISPLAY (DEV
ICE TWO) */

```



```

686      DCL (A, TAIL) BYTE;
687      DCL (X1, Y1, X2, Y2, B, C) ADDRESS;
688      DCL (ZINDEX, ZSHIPINDEX) BYTE;
689      ZINDEX=0;
690      ZSHIPINDEX=0;
691      DO WHILE ZINDEX < 3 ;
692          IF (SHIP$PLOT(ZINDEX).COUNT) > 0 THEN
693              DO WHILE ZSHIPINDEX < SHIP$PLOT(ZINDEX).COUNT;
694                  IF ZINDEX=0 THEN
695                      CALL DRAW$FRIEND$SYMBOL$2(
696                          SHIP$PLOT(0).X$BOW(ZSHIPINDEX),
697                          SHIP$PLOT(0).Y$BOW(ZSHIPINDEX), 1);
698                      ELSE
699                          IF ZINDEX=1 THEN
700                              CALL DRAW$HOSTILE$SYMBOL$2(
701                                  SHIP$PLOT(1).X$BOW(ZSHIPINDEX),
702                                  SHIP$PLOT(1).Y$BOW(ZSHIPINDEX), 1);
703                              ELSE
704                                  IF ZINDEX=2 THEN
705                                      CALL DRAW$UNKNOWN$SYMBOL$2(
706                                          SHIP$PLOT(2).X$BOW(ZSHIPINDEX),

```



```
SHIP$PLOT(2).Y$BOW(ZSHIPINDEX),1);
```

```
/* DRAW TAIL FOR SHIP */
IF TAIL THEN
DO;
X1=SHIP$PLOT(ZINDEX).X$BOW(ZSHIPINDEX);
Y1=SHIP$PLOT(ZINDEX).Y$BOW(ZSHIPINDEX);
IF NOT (( X1<7) OR (X1>504) OR (Y1<7) OR (Y1>504))
THEN
IF SHIP$PLOT(ZINDEX+3).COUNT > 0 THEN
DO;
IF SHIP$PLOT(ZINDEX+6).COUNT <=0 THEN A=3;
ELSE IF SHIP$PLOT(ZINDEX+9).COUNT <=0 THEN A=6;
ELSE IF SHIP$PLOT(ZINDEX+12).COUNT <=0 THEN A=9;
ELSE A=12;
X2=SHIP$PLOT(ZINDEX+A).X$BOW(ZSHIPINDEX);
Y2=SHIP$PLOT(ZINDEX+A).Y$BOW(ZSHIPINDEX);
B=X1 + (X2-X1) + (X2-X1) + (X2-X1);
```



```
717 6 = C=Y1 + (Y2-Y1) + (Y2-Y1) + (Y2-Y1);
718 6 = CALL START$VECTOR$SOLID$2(X1,Y1);
719 6 = CALL STOP$VECTOR$SOLID$2(B,C);
720 6 = END; /* IF SHIP$PLOT(ZINDEX+3) */
721 5 = END; /* IF TAIL */
=
=
=
=
=
=
=
=
=
/* LABEL CONTACT SYMBOL */
722 4 = IF SHIP$PLOT(ZINDEX).X$BOW(ZSHIPINDEX)<=486 THEN
723 4 = CALL WRITE$CONTACT$ID$2(SHIP$PLOT(ZINDEX).X$BOW(ZSH
IPINDEX)+11,
EX);
=
=
724 4 = ELSE
725 4 = CALL WRITE$CONTACT$ID$2(SHIP$PLOT(ZINDEX).X$BOW(ZSH
IPINDEX)-18,
EX);
=
=
726 4 = ZSHIPINDEX=ZSHIPINDEX+1;
726 4 = END; /* DO WHILE ZSHIPINDEX */
```



```

727 3 = ZSHIPINDEX=0;
728 3 = ZINDEX=ZINDEX+1;
729 3 = END; /* DO WHILE ZINDEX < 3 */
730 2 = END; /* DRAW$SHIP$2 */
      =
      = /* END; GRAPH2$MODULE */

```

```

731 1      END; /* GRAPH1 MODULE */

```

351

MODULE INFORMATION:

```

CODE AREA SIZE      = 194FH      6479D
VARIABLE AREA SIZE = 0D8BH      3467D
MAXIMUM STACK SIZE = 0008H      8D
1233 LINES READ
0 PROGRAM ERROR(S)

```


PL/M-80 COMPILER MODULE GRAPH1.MOD
979 PAGE 40

MAY 1

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PDPMODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:PDP.MOD NOOBJECT PAGEWIDTH(82) PAGEL
ENGTH(24) DATE
 -(MAY 1979) TITLE('MODULE PDP.MOD')

1 PDP\$MODULE:
 DO;

\$INCLUDE(:F1:MP.DEC)

/* DECLARATIONS: *****/

2 1 DECLARE LIT LITERALLY 'LITERALLY';
3 1 DECLARE DCL LIT 'DECLARE';


```

= =
4 1      DCL TOP LIT '0F800H'; /* TOP OF FREE SPACE */
5 1      DCL BASE LIT '0F000H'; /* BASE OF PDP BUFFER */
6 1      DCL RECEIVE$PRINT BYTE EXTERNAL;

```

/* PROGRAM CONSTANTS: *****/

```

7 1      DCL TRUE LIT '0FFH';
8 1      DCL FALSE LIT '000H';
9 1      DCL FOREVER LIT 'WHILE 1';
10 1     DCL RCVD LIT '2EH';
11 1     DCL NEUTRAL LIT '0';
12 1     DCL RECEIVE LIT '1';
13 1     DCL TRANSMIT LIT '2';
14 1     DCL ACK LIT '06H';

```



```
21 1 = = = = = = = = = = = = = = = =  
      WRITE$STATUS,  
      DELETE$STATUS,  
      READ$STATUS,  
      OPEN$STATUS,  
      PDP$BUF$PTR,  
      CLOSE$STATUS,  
      CONSOL$STATUS,  
      BYTE$COUNT,  
      COUNT$TEMP,  
      AFT$IN ) ADDRESS ;  
      DCL ( MISSINGDATA,  
      STOP$FLAG,  
      SENT$ACK,  
      ALARM,  
      D1,D2,D3,  
      CHAR$TWO,  
      CHAR,  
      STATE,  
      PREVCHAR,  
      EOFILE,  
      SKIP$FIVE,  
      =
```


	=	ECHOCHAR,	
	=	SAVECHAR) BYTE ;	
22	1	DCL CRTCHO BYTE INITIAL (0);	
23	1	DCL PDPECHO BYTE INITIAL (0);	

/* SPECIAL CHARACTERS: *****/

24	1	=	=	=	=	=	=	=	=	=	=	=	=	=	DCL	RUBOUT	LIT	'7FH'	/*	PERCENT	*/
															PROMPT	LIT	'25H'				
															CONTROL\$W	LIT	'17H'				
															CONTROL\$N	LIT	'4EH'				
															CONTROL\$R	LIT	'12H'				
															CONTROL\$T	LIT	'14H'				
															CONTROL\$Z	LIT	'1AH'				
															CONTROL\$C	LIT	'03H'				
															CR	LIT	'0DH'				
															LF	LIT	'0AH'				

BELL LIT '07H';

/*ISIS-II SYSTEM CALLS: *****/

```

25      1      OPEN:
26      2      PROCEDURE (AFT, FILE, ACCESS, MODE, STATUS ) EXTER
27      2      NAL;
28      1      DECLARE (AFT, FILE, ACCESS, MODE, STATUS ) AD
29      2      DRESS;
30      2      END OPEN;

28      1      CLOSE:
29      2      PROCEDURE(AFT, STATUS) EXTERNAL;
30      2      DCL (AFT, STATUS) ADDRESS;
30      2      END CLOSE;

```


31	1	=	READ:	PROCEDURE(AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTE
		=		
			RNAL;	
32	2	=		DCL(AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
33	2	=	END READ;	
		=		
		=	WRITE:	
34	1	=		PROCEDURE(AFT,BUFFER,COUNT,STATUS) EXTERNAL;
		=		DCL(AFT,BUFFER,COUNT,STATUS) ADDRESS;
35	2	=		
36	2	=	END WRITE;	
		=		
		=		
		=	EXIT:	
37	1	=		PROCEDURE EXTERNAL;
		=		DCL STATUS ADDRESS;
38	2	=		
39	2	=	END EXIT;	
		=		
		=		
		=	CONSOL:	
40	1	=		


```
41      =      PROCEDURE(INFILE,OUTFILE,STATUS) EXTERNAL;  
42      2      DCL (INFILE,OUTFILE,STATUS) ADDRESS;  
      =      END CONSOL;  
      =  
      =  
      =  
43      1      DELETE:  
      =      PROCEDURE(FILE,STATUS) EXTERNAL;  
44      2      DCL(FILE,STATUS) ADDRESS;  
45      2      END DELETE;  
      =  
      =  
      =  
46      1      ERROR:  
      =      PROCEDURE (ERRNUM) EXTERNAL;  
47      2      DCL (ERRNUM) ADDRESS;  
48      2      END ERROR;  
      =  
      =  
49      1      RENAME:  
      =      PROCEDURE(OLDFILE,NEWFILE,STATUS) EXTERNAL;  
      =
```


50 2 = DCL(OLDFILE,NEWFILE,STATUS) ADDRESS;
51 2 = END RENAME;

/* PROCEDURES:*****//

52 1 SET\$TTY\$2400: PROCEDURE;

53 2 /* SET TTY BAUD RATE TO 2400 */
54 2 OUTPUT(245)=40H;
55 2 OUTPUT(245)=4FH;
56 2 OUTPUT(245)=37H;
END;

= = = = =

```

57      1      OUTPUT$STATUS$CRT: PROCEDURE BYTE;
      2      /* TRUE IF DATA OUTPUT LINE TO CRT READY */
58      2      RETURN ROR (INPUT(247),2);
59      2      END;
```

60 1

OUTPUT\$STATUS\$PDP: PROCEDURE BYTE;

```

61      2      /* TRUE IF DATA OUTPUT LINE TO PDP READY */
62      2      RETURN ROR(INPUT(245),2);
      2      END;
```



```
63 1  =  =
    =  =  =
64 2  /* TRUE IF DATA INPUT LINE FROM CRT READY */
65 2  RETURN ROR(INPUT(247),1);
    END;

66 1  INPUT$STATUS$PDP: PROCEDURE BYTE ;
    =  =  =
67 2  /* TRUE IF DATA INPUT LINE FROM PDP READY */
68 2  RETURN ROR(INPUT(245),1) ;
    END;

69 1  SEND$CHAR$CRT: PROCEDURE (CHAR) ;
    =  =  =
70 2  /* PRINT A CHARACTER TO THE CRT */
71 2  DCL CHAR BYTE;
    DO WHILE NOT OUTPUT$STATUS$CRT;
```



```

72 3  =
73 2  =
74 2  =
    =
    =
    =
75 1  = SEND$CHAR$PDP: PROCEDURE (CHAR);
    =
    =
    =
76 2  /* SEND A CHARACTER TO THE PDP */
77 2  DCL CHAR BYTE;
78 3  DO WHILE NOT OUTPUT$STATUS$PDP;
79 2  END;
80 2  OUTPUT(244)= CHAR;
    =
    =
    =
    =
    =
    =
    =
    =
    =
81 1  /* CRLF: */
    CRLF: PROCEDURE ;
    /* SEND CARRIAGE RETURN AND LINE FEED

```



```

82      2      =      CALL SEND$CHAR$CRT(CR);
83      2      =      CALL SEND$CHAR$CRT(LF);
84      2      =      END;
      =
      =
      =
85      1      =      SEND$STRING$CRT: PROCEDURE (STRING$ADDRESS);
      =
      =      /* SEND MESSAGE AT STRING$ADDRESS UNTIL '$' IS
      =      DETECTED */
      =      DCL STRING$ADDRESS ADDRESS;
86      2      =      DCL TEMPCHAR BASED STRING$ADDRESS BYTE;
87      2      =      DO WHILE TEMPCHAR <> '$';
88      2      =      CALL SEND$CHAR$CRT(TEMPCHAR);
89      3      =      STRING$ADDRESS=STRING$ADDRESS + 1;
90      3      =      END;
91      3      =      END;
92      2      =
      =
      =
93      1      =      PRINT$TO$CRT: PROCEDURE (STRING$ADDRESS) ;

```



```

106 1 READ$CHAR$PDP:PROCEDURE BYTE;
    =
    =
    =
107 2 /* READ A CHARACTER FROM THE PDP */
108 2 DCL CHAR BYTE;
109 2 CHAR= INPUT (244) ;
110 2 RETURN CHAR;
    =
    =
    =
    =
    =
111 1 GET$CHAR$CRT$BUF: PROCEDURE BYTE;
    =
    =
    =
112 2 /* GET A CHARACTER FROM THE CRT BUFFER */
    =
    =
    =
113 2 DCL CHAR BYTE;
114 2 /* GET FIRST CHARACTER */
    =
    =
    =
115 2 CHAR =CRT$BUFFER(CRT$BUF$FIRST);
116 2 IF (CRT$BUF$FIRST:=CRT$BUF$FIRST+1)>LAST(CRT$BU
    =
    =
    =
    =
117 2 THEN CRT$BUF$FIRST=0; /* WRAP AROUND */
118 2 CRT$BUF$NUMBER=CRT$BUF$NUMBER - 1;
    =
    =
    =
    =
119 2 RETURN CHAR;
120 2 END;

```



```

=
=
=
119 1 GET$CHAR$PDP$BUF: PROCEDURE BYTE;
=
=
120 2 /* GET A CHAR FROM THE PDP BUFFER */
=
=
121 2 DCL CHAR BYTE;
=
=
122 2 /* GET FIRST CHARACTER */
=
=
PDP$BUF CHAR=PDP$BUFFER(PDP$BUF$FIRST);
122 2 IF (PDP$BUF$FIRST=PDP$BUF$FIRST + 1)>LAST$LOC$
=
=
124 2 THEN PDP$BUF$FIRST=0;
=
=
125 2 PDP$BUF$NUMBER=PDP$BUF$NUMBER - 1;
=
=
126 2 RETURN CHAR;
=
=
=
=
=
127 1 PUT$CHAR$CRT$BUF: PROCEDURE (CHAR);
=
=
=
=
128 2 /* A CHAR IS PUT IN THE CRT BUFFER */
=
=
DCL CHAR BYTE;
```



```
129 2 = IF(CRT$BUF$LAST:=CRT$BUF$LAST + 1)> LAST(CRT$BU  
FFER)  
131 2 = THEN CRT$BUF$LAST=0;  
132 2 = IF CRT$BUF$NUMBER= 0 THEN  
133 2 = CRT$BUF$LAST=CRT$BUF$FIRST;  
134 2 = CRT$BUFFER(CRT$BUF$LAST)=CHAR;  
135 2 = CRT$BUF$NUMBER=CRT$BUF$NUMBER+1;  
136 2 = ALARM=0; /* SET FOR NO ALARM FOR NEXT PASS */  
= = = = =  
137 1 = PUT$CHAR$PDP$BUF: PROCEDURE(CHAR);  
= = = = =  
138 2 = /* PUT CHARACTER IN PDP BUFFER */  
139 2 = DCL CHAR BYTE;  
= IF(PDP$BUF$LAST:=PDP$BUF$LAST + 1)>LAST$LOC$PDP$  
BUF  
= = = = =  
141 2 = THEN PDP$BUF$LAST=0;  
142 2 = IF(PDP$BUF$NUMBER=0)THEN  
143 2 = PDP$BUF$LAST=PDP$BUF$FIRST;  
144 2 = PDP$BUFFER(PDP$BUF$LAST)=CHAR;  
= PDP$BUF$NUMBER=PDP$BUF$NUMBER+1;  
= = = = =
```



```

145 2 = = = = =
      END;

146 1 = = = = =
      CRT$BUF$FULL: PROCEDURE BYTE;

147 2 = = = = =
      /* CHECK IF CRT BUFFER OVERFLOW */
148 2 = = = = =
      RETURN CRT$BUF$NUMBER=LENGTH(CRT$BUFFER);
      END;

149 1 = = = = =
      PDP$BUF$FULL: PROCEDURE BYTE;

150 2 = = = = =
      /* CHECK FOR PDP BUFFER OVERFLOW */
151 2 = = = = =
      RETURN PDP$BUF$NUMBER=LENGTH$PDP$BUF;
      END;

152 1 = = = = =
      PRINT$HEX$NUMBER: PROCEDURE (CHAR);

```



```

=
=
153      2      /* PRINT HEXADECIMAL COUNTER */
154      2      DCL CHAR BYTE;
156      2      IF CHAR>9 THEN CALL SEND$CHAR$CRT(CHAR-10+'A');
157      2      ELSE CALL SEND$CHAR$CRT(CHAR+'0');
      END;

=
=
158      1      FORMAT$HEX :PROCEDURE (CHAR);
      /* FORMAT DECIMAL NUMBER FOR HEXADECIMAL OUTPUT */
159      2      DCL CHAR BYTE;
160      2      CALL PRINT$HEX$NUMBER(SHR(CHAR,4));
161      2      CALL PRINT$HEX$NUMBER(CHAR AND 0FH);
162      2      END;

=
=
163      1      PRINT$CHAR$COUNT: PROCEDURE ;

=
=
      /* PRINT DECIMAL COUNTER IN HEXADECIMAL FORMAT */
=
```


164	2	=	CALL CRLF;
165	2	=	CALL SEND\$STRING\$CRT(.('CHARACTER COUNT:\$'));
166	2	=	IF D1 <> 0 THEN CALL FORMAT\$HEX(D1);
168	2	=	IF D2 <> 0 THEN CALL FORMAT\$HEX(D2);
170	2	=	CALL FORMAT\$HEX(D3);
171	2	=	END;
		=	
		=	
		=	
172	1	=	INIT\$CHAR\$COUNT: PROCEDURE;
		=	
		=	/* CLEAR DECIMAL COUNTER */
173	2	=	D3,D2,D1=0;
174	2	=	END;
		=	
		=	
		=	
175	1	=	COUNT\$CHAR: PROCEDURE ;
		=	
		=	/* INCREMENT DECIMAL COUNTER */
176	2	=	D3=DEC(D3+1);


```

177      D2=DEC(D2 PLUS 0);
178      D1=DEC(D1 PLUS 0);
179      END;
      =
      =
      =
      =
      =
      =
      =
180      INIT$NEUTRAL$STATE: PROCEDURE;
      =
      =
      =
181      /* INITIALIZE NEUTRAL STATE */
182      STATE=NEUTRAL;
      =
      =
      =
      =
183      CALL CRLF;
184      CALL SEND$CHAR$PDP(CR);
185      END;
      =
      =
      =
      =
      =
      =
      =
186      INIT$RECEIVE$STATE: PROCEDURE ;
      =
      =
      =
      =
187      /* INITIALIZE RECEIVING STATE */
      PDP$BUF$FIRST=0;

```



```
188 =
189 2
190 2
191 2
192 2
193 2
194 3
195 3
196 3
197 2
198 2
199 2
200 2
201 2
202 1
203 2

PDP$BUF$LAST=0;
PDP$BUF$NUMBER=0;
PREVCHAR=',';
STATE=RECEIVE;
IF RECEIVE$PRINT=TRUE THEN
DO;
CALL SEND$STRING$CRT(('.SYSTEM IN RECEIVE STATE:
$'));
CALL CRLF;
END; /* IF RECEIVE$PRINT */
CALL INIT$CHAR$COUNT;
SKIP$FIVE = 0;
CALL SEND$CHAR$PDP('&');
CALL SEND$CHAR$PDP(CR);
END;

INIT$TRANSMIT$STATE: PROCEDURE;

/* INITIALIZE TRANSMIT STATE */
EOFIL=0;
```



```

204      ECHOCHAR=1;
205      SENT$ACK = FALSE;
206      SAVECHAR=',';
207      STATE=TRANSMIT;
208      CALL PRINT$TO$CRT(,('SYSTEM IN TRANSMIT STATE:
    $'));
209      CALL CRLF;
210      STOP$FLAG=FALSE;
211      CALL INIT$CHAR$COUNT;
212      END;

213      BREAK$STATE: PROCEDURE ;

    /* INTERRUPT THE PDP */
    DCL I BYTE;
    DO I = 1 TO 3;
        CALL SEND$CHAR$PDP(RUBOUT);
    END;
    END;

```



```

=
=
=
219 1 = END$R : PROCEDURE ;
=
=
STATE */
220 2 = /* TERMINATE RECEIVE STATE ,AND RETURN TO NEUTRAL
Y OPERATOR.$' ) ;
CALL PRINT$TO$CRT.( 'RECEIVE STATE TERMINATED B
-
221 2 = CALL PRINT$TO$CRT.( 'NO FILE CREATED AT FLOPPY
DISK.$' ) );
=
222 2 = /* DISCARD PDP BUFFER CONTENTS */
223 2 = PDP$BUF$FIRST=0;
224 2 = PDP$BUF$LAST=0;
= PDP$BUF$NUMBER=0;
/* SEND BREAK SIGNAL TO THE PDP */
= CALL BREAK$STATE;
= CALL CLOSE(AFT$IN,.CLOSE$STATUS);
= CALL DELETE(.FILENAME,DELETE$STATUS);
228 2 = END;
=
=
=

```



```

229 1 =      END$T: PROCEDURE ;
    =
    =
    =
TE */
230 2 =      CALL PRINT$TO$CRT(.( 'TRANSMIT STATE TERMINATED
BY OPERATOR.$'
    -
    ));
231 2 =      CALL PRINT$TO$CHK(.( 'PARTIAL FILE CREATED AT PD
P.$' ));
    =
232 2 =      /* SEND BREAK SIGNAL TO PDP */
233 2 =      CALL SEND$CHAR$PDP(ACK);
234 2 =      SENT$ACK = TRUE;
235 2 =      CALL CLOSE (AFT$IN,.CLOSE$STATUS);
    =      CALL CONSOL (.( ':CI:$' ),.( ':VO:$' ),..CONSOL$
STATUS);
236 2 =      END;
    =
    =
    =
    =
237 1 =      WRITE$RECORD$TO$DISK:PROCEDURE;
    =
238 2 =      /* WRITE ONE RECORD FROM PDPBUF TO DISK */
    =      CALL WRITE(AFT$IN,.PDP$BUFFER(PDP$BUF$PTR),128,
.WRITE$STATUS)

```



```

239 2 = - ;
    E$STATUS);
240 2 =
241 2 =
242 2 =
243 3 =
244 3 =
245 2 =
246 2 =

247 1 =
    WRITE$PDP$BUFFER: PROCEDURE;

248 2 = /* WRITE ENTIRE PDP BUFFER TO DISK */
249 2 =     PDP$BUF$PTR = 0002H;
    PDP$BUF$NUMBER = PDP$BUF$NUMBER -3;
    /* PAD PDP BUFFER UNTIL PDP$BUF$NUMBER IS A MUL-
        TIPL OF 128 */
250 2 =     DO WHILE ((PDP$BUF$NUMBER - 2) AND 0177Q) <>
    0;

```



```
251 3 = CALL PUT$CHAR$PDP$BUF(' ');
252 3 = END;
253 2 = CALL OPEN(.AFT$IN,.FILE$NAME,2,0,.OPEN$STAT
US);
254 2 = DO WHILE PDP$BUF$NUMBER <> 2;
/* WRITE NEXT 128 BYTE RECORD TO DISK */
255 3 = CALL WRITE$RECORD$TO$DISK;
256 3 = END;
257 2 = CALL CLOSE (AFT$IN,.CLOSE$STATUS);
258 2 = PDP$BUF$PTR = 0002H;
259 2 = PDP$BUF$LAST,PDP$BUF$FIRST = 0;
260 2 = END;

/* REBOOT: */
261 1 = REBOOT: PROCEDURE;

/* GO BACK TO MDS OPERATING SYSTEM : ISIS-11 */
262 2 = IF MISSINGDATA THEN CALL PRINT$TO$CRT(
CHAR TRANSMIT.('NEUTRAL STATE: SYSTEM MAY NOT HAVE RECEIVED ALL PDP
```



```

-
=
=
264      TED $')');
        CALL PRINT$TO$CART(
          .('NEUTRAL STATE AND REBOOTING TO ISIS-11 $'))
;
265      CALL CONSOL(.('CI:$'),.(':VO:$'),.CONSOL$STATU
S);
266      END;
=
=
=
267      PDP$STRUCTURE: PROCEDURE PUBLIC ;

/* THIS PROCEDURE IS RESPONSIBLE FOR THE MDS-PDP INTE
RFACE */
$INCLUDE(:F1:MP.COD)

/* MAIN ROUTINE: *****/
=
=
=
=
=
=
=
=
=
ACTION **/
/* BEGIN TESTING PORTS , AND SELECT THE APPROPRIATE
```



```

268 2  =  /* SET THE PDP BUFFER PARAMETERS */
269 2  =  LENGTH$PDP$BUF=2000;
      =  LAST$LOC$PDP$BUF=LENGTH$PDP$BUF-1;
      =  /* NOW, THE POINTERS OF THE BUFFERS OF THE CRT AND
THE
270 2  =  PDP ARE INITIALIZED */
      =  PDP$BUF$FIRST,
      =  PDP$BUF$LAST,
      =  PDP$BUF$NUMBER,
      =  CRT$BUF$FIRST,
      =  CRT$BUF$LAST,
      =  CRT$BUF$NUMBER=0;
      =
      =
      =
      =  /* INITIALIZE SYSTEM BY ENTERING THE RECEIVE STATE
      : */
271 2  =  MISSINGDATA=FALSE;
272 2  =  ALARM=FALSE;
273 2  =  CALL INIT$RECEIVE$STATE;
      =
      =
      =  /* BEGIN INFINITE LOOP:  *****/

```



```

328 6 = CALL REBOOT;
329 6 = RETURN;
330 6 = END; /* PDP BUFFER FULL */
      /* BUFFER NOT FULL */
      /* GET CHARACTER FROM PDP */
331 5 = CHAR=READ$CHAR$PDP;
      /* PDP PROMPTING : TERMINATE RECEPTION */
332 5 = IF ((PREVCHAR=LF) OR (PREVCHAR=CR)) AND CHAR=
PROMPT
      THEN
333 5 = DO; /* PROMPTING */
334 6 = IF RECEIVE$PRINT=TRUE THEN
335 6 = DO;
336 7 = CALL SEND$STRING$CRT(.( 'PDP PROMPTING:END
OF RECEPTION.$
      '));
337 7 = CALL CRLF;
338 7 = CALL SEND$STRING$CRT(
      .('PDP BUFFER/FILE WRITTEN TO STRUCTUR
E.$'));
      /* PRINT NUMBER OF CHARACTERS TRANSMITTED
      FROM PDP BUFFER TO STRUCTURE */
339 7 = CALL CRLF;
340 7 = CALL PRINT$CHAR$COUNT;

```



```
341 7 = =
RE.$' ));
342 7 = =
343 6 = =
344 6 = =
345 5 = =
346 5 = =
347 6 = =
348 6 = =
349 6 = =
350 6 = =
351 6 = =
352 5 = =
353 6 = =
354 6 = =

        CALL SEND$STRING$CRT(
        .(' BYTES TRANSMITTED FROM PDP TO STRUCTU

        /* RECEPTION COMPLETE: RETURN TO LOOP */
END; /* IF RECEIVE$PRINT */
        RETURN;
        END; /* PROMPTING */
        ELSE
            IF CHAR=END$OF$BLOCK THEN
                DO;
                    CALL WRITE$PDP$BUFFER;
                    CALL PRINT$TO$CRT(
                        .('RECEIVE STATE: RECEIVED BLOCK.$' ));
                    /* CRT ACKNOWLEDGE RECEPTION FROM PDP */
                    CALL PUT$CHAR$CRT$BUF(RCVD);
                    CALL PUT$CHAR$CRT$BUF(CR);
                END;
            ELSE /* NOT END OF RECEPTION-DATA */
                DO;
                    PREV CHAR=CHAR;
                    CALL PUT$CHAR$PDP$BUF(CHAR);
                    /* INCREMENT NUMBER OF CHARS RECEIVED */
```



```

355 6 = IF (SKIP$FIVE < 4) THEN
356 6 = DO;
357 7 = SKIP$FIVE = SKIP$FIVE + 1;
358 7 = END;
      ELSE
359 6 = DO;
360 7 = CALL COUNT$CHAR;
361 7 = END;
      END;
362 6 =
363 5 = END; /* END RECEIVING */
      ELSE /* NEUTRAL OR TRANSMIT STATE */
364 4 = DO;
      /* PDP BUFFER FULL */
365 5 = IF PDP$BUF$FULL THEN MISSINGDATA=1;
      /* BUFFER NOT FULL */
      ELSE
367 5 = DO;
368 6 = CHAR=READ$CHAR$PDP;
369 6 = IF SAVECHAR=CHAR THEN ECHOCHAR=1;
371 6 = CALL PUT$CHAR$PDP$BUF(CHAR);
372 6 = END;

```



```

383 3      /*FOURTH CASE: SEND DATA TO PDP */
384 3      IF OUTPUT$STATUS$PDP THEN
385 4      DO; /* SEND DATA TO PDP */
386 4      IF STATE <> TRANSMIT THEN
/      DO; /* CHECK IF CRT HAS SOMETHING FOR THE PDP *
387 5      IF CRT$BUF$NUMBER > 0 THEN
388 5      CALL SEND$CHAR$PDP(GET$CHAR$CRT$BUF);
389 5      END; /* SEND DATA TO PDP */
390 4      ELSE /* TRANSMITTING DATA */
      DO;
391 5      CALL OPEN (.AFT$IN,.FILE$NAME,1,0,.OPE
N$STATUS);
392 5      BYTE$COUNT=1;
393 5      CALL SEND$CHAR$PDP(CR);
394 5      DO WHILE (BYTE$COUNT<>0 AND STOP$FLAG = FALSE)
;
395 6      IF INPUT$STATUS$CRT THEN
396 6      DO;
397 7      CHAR$TWO=READ$CRT$CHAR;
398 7      IF CHAR$TWO=CONTROL$C THEN
399 7      DO;
```



```

400 8 =
401 8 =
402 8 =
403 7 =
404 6 =
405 6 =
406 7 =
407 7 =
408 8 =
.BYTE$COUNT,
=
409 8 =
,.CONSOL$STATU
S);
410 8 =
T,.WRITE$STATU
S);
411 8 =
.CONSOL$STATUS
);
412 8 =
T,.WRITE$STATU
S);
413 8 =
414 8 =
<128)) THEN
415 8 =
CALL END$T;
STOP$FLAG = TRUE;
END;
END;
IF INPUT$STATUS$PDP THEN
DO;
IF (CHAR:=READ$CHAR$PDP) = '+' THEN
DO;
CALL READ(AFT$IN,.PDP$BUFFER,128,
.READ$STATUS);
CALL CONSOL (('.'CI:$'),.('TO:$'))
CALL WRITE(0,.PDP$BUFFER,BYTE$COUN
CALL CONSOL(('.'CI:$'),.('VO:$')),
CALL WRITE(0,.PDP$BUFFER,BYTE$COUN
CHAR=' ';
IF ((BYTE$COUNT>0) AND (BYTE$COUNT
DO;

```



```
416 =  
417 =  
418 =  
419 =  
420 =  
    CALL SEND$CHAR$PDP(ACK);  
    SENT$ACK=TRUE;  
    END;  
    COUNT$TEMP=0;  
    DO WHILE (BYTE$COUNT-COUNT$TEMP)<>  
0;  
421 =  
422 =  
423 =  
424 =  
425 =  
    CALL COUNT$CHAR;  
    COUNT$TEMP=COUNT$TEMP+1;  
    END;  
    END;  
    END;  
426 =  
427 =  
428 =  
    END;  
    IF(SENT$ACK = FALSE) THEN  
    DO;  
429 =  
430 =  
431 =  
432 =  
433 =  
        CALL SEND$CHAR$PDP(ACK);  
    END;  
    CALL CLOSE(AFT$IN,.CLOSE$STATUS);  
    CALL PRINT$CHAR$COUNT;  
    CALL SEND$STRING$CRT(  
    .(' BYTES TRANSMITTED FROM FLOPPY DISK TO PDP.  
    $'));  
434 =  
435 =  
    CALL CRLF;  
    CALL PRINT$TO$CRT(
```



```

436 = .
437 =
438 =
439 =
440 =
      .('LEAVING TRANSMIT STATE; ENTER LOOP.$`')));
      RETURN;
      END; /*TRANSMIT */
      END; /* END FOURTH CASE */
      END; /* DO FOREVER */
      END; /* PDP STRUCTURE */
441 1      END; /* PDP$MODULE */

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 09FBH 2555D
VARIABLE AREA SIZE = 08F3H 2291D
MAXIMUM STACK SIZE = 000AH 10D
833 LINES READ
0 PROGRAM ERROR(S)

```


PL/M-80 COMPILER MODULE PDP.MOD
979 PAGE 42

MAY 1

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PLASMA MODULE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:PLASMA.MOD NOOBJECT PAGEWIDTH(82) PA
GELENGTH(24) D
-ATE(MAY 1979) TITLE('MODULE PLASMA.MOD')

```

1      PLASMA$MODULE:DO;
      /* PLASMA MODULE : PRIMITIVES AND EXTERNALS FOR
      THE TWO PLASMA DEVICES. */
2      1      DECLARE LIT LITERALLY 'LITERALLY';
3      1      DECLARE DCL LIT 'DECLARE';

      $INCLUDE(:F1:PLAEXT.ONE)
=

```



```

/* PLASMA PRIMITIVES 1 */

=
=
=
=
=
4    1    SET$STATUS$PLASMA:PROCEDURE (STATUS) EXTERNAL;
5    2    DCL STATUS BYTE;
6    2    END;

=
=
7    1    PLASMA$WRITE:PROCEDURE (CHAR) EXTERNAL;
8    2    DCL CHAR BYTE;
9    2    END;

=
=
10   1    CLEAR$PLASMA:PROCEDURE EXTERNAL;
11   2    END;

=
=
12   1    PLASMA$WRITE$VECTOR:PROCEDURE (A) EXTERNAL;
13   2    DCL A ADDRESS,

```



```

=
=
14 2 = END;
=
15 1 = PLASMA$PRINT$STRING:PROCEDURE (COLUMN,ROW,POINTER) EXT
ERNAL;
16 2 = DCL POINTER ADDRESS,
= BUFFER BASED POINTER (1) BYTE,
= (COLUMN,ROW,COUNT) BYTE;
17 2 = END;
=
=
18 1 = INITIALIZE$PLASMA:PROCEDURE EXTERNAL;
19 2 = DCL BUFFER (*) BYTE DATA ('ON LINE.$$');
20 2 = END;
=
=
21 1 = SET$VECTOR:PROCEDURE (X,Y,POINTER) EXTERNAL;
22 2 = DCL (X,Y,POINTER) ADDRESS,
= VECTOR BASED POINTER (4) BYTE;
23 2 = END;
```



```

=
=
24 1 = START$VECTOR$SOLID:PROCEDURE (X,Y) EXTERNAL;
25 2 = DCL (X,Y) ADDRESS,
26 2 = VECTOR (4) BYTE;
27 2 = END;
=
=
27 1 = STOP$VECTOR$SOLID:PROCEDURE (X,Y) EXTERNAL;
28 2 = DCL (X,Y) ADDRESS,
29 2 = VECTOR (4) BYTE;
30 2 = END;
=
=
30 1 = START$VECTOR$DASH:PROCEDURE (X,Y) EXTERNAL;
31 2 = DCL (X,Y) ADDRESS,
32 2 = VECTOR (4) BYTE;
33 2 = END;
=
=
33 1 = STOP$VECTOR$DASH:PROCEDURE (X,Y) EXTERNAL;

```



```

34 2 = DCL (X,Y) ADDRESS,  
35 2 = VECTOR (4) BYTE;  
    = END;  
    =  
    =  
36 1 = GRAPHIC$DESIG:PROCEDURE (X,Y,DESIG) EXTERNAL;  
37 2 = DCL (X,Y,DESIG) ADDRESS,  
    = VALUE BASED DESIG ADDRESS ,  
    = BUFFER (6) BYTE,  
    = (ROW,COLUMN) BYTE;  
38 2 = END;  
    =  
    =  
39 1 = START$ERASE$VECTOR:PROCEDURE (X,Y) EXTERNAL;  
40 2 = DCL (X,Y) ADDRESS,  
41 2 = VECTOR (4) BYTE;  
    = END;  
    =  
    =  
42 1 = STOP$ERASE$VECTOR:PROCEDURE(X,Y) EXTERNAL;
```



```

43 2 = = DCL (X,Y) ADDRESS,
44 2 = = VECTOR (4) BYTE;
45 1 = = END;
46 2 = = START$ERASE$DASH:PROCEDURE (X,Y) EXTERNAL;
47 2 = = DCL (X,Y) ADDRESS,
48 1 = = VECTOR (4) BYTE;
49 2 = = END;
50 2 = = STOP$ERASE$DASH:PROCEDURE (X,Y) EXTERNAL;
/* END EXTERNAL PLASMA PRIMITIVES 1 */

```


51	1	=	
52	2	=	
53	2	=	
54	1	=	
55	2	=	
56	2	=	
57	1	=	

```

$INCLUDE(:F1:PLAEXT.TWO)

/* PLASMA PRIMITIVES 2 */

SET$STATUS$PLASMA$2:PROCEDURE (STATUS) EXTERNAL;
  DCL STATUS BYTE;
  END;

PLASMA$WRITE$2:PROCEDURE (CHAR) EXTERNAL;
  DCL CHAR BYTE;
  END;

CLEAR$PLASMA$2:PROCEDURE EXTERNAL;

```



```
58 2 = END;
    =
    =
59 1 = PLASMA$WRITE$VECTOR$2:PROCEDURE (A) EXTERNAL;
60 2 =   DCL A ADDRESS,
    =   VECTOR BASED A (4) BYTE,
    =   VPTR BYTE;
    =
61 2 = END;
    =
    =
62 1 = PLASMA$PRINT$STRING$2:PROCEDURE (COLUMN,ROW,POINTER) E
XTERNAL;
63 2 =   DCL POINTER ADDRESS,
    =   BUFFER BASED POINTER (1) BYTE,
    =   (COLUMN,ROW,COUNT) BYTE;
    =
64 2 = END;
    =
    =
65 1 = INITIALIZE$PLASMA$2:PROCEDURE EXTERNAL;
66 2 =   DCL BUFFER (*) BYTE DATA ('ON LINE.$$');
67 2 = END;
    =
```


MAY 1

```

68      1      =      SET$VECTOR$2:PROCEDURE (X,Y,POINTER) EXTERNAL;
69      2      =           DCL (X,Y,POINTER) ADDRESS,
70      2      =           VECTOR BASED POINTER (4) BYTE;
71      2      =           END;
72      1      =      START$VECTOR$SOLID$2:PROCEDURE (X,Y) EXTERNAL;
73      2      =           DCL (X,Y) ADDRESS,
74      2      =           VECTOR (4) BYTE;
75      2      =           END;
76      1      =      STOP$VECTOR$SOLID$2:PROCEDURE (X,Y) EXTERNAL;
77      2      =           DCL (X,Y) ADDRESS,
78      2      =           VECTOR (4) BYTE;
79      2      =           END;
80      1      =      START$VECTOR$DASH$2:PROCEDURE (X,Y) EXTERNAL;
81      2      =           DCL (X,Y) ADDRESS,
82      2      =           VECTOR (4) BYTE;
83      2      =           END;

```



```

79      2      =      VECTOR(4) BYTE;
          =      END;
          =
          =
80      1      =      STOP$VECTOR$DASH$2:PROCEDURE (X,Y) EXTERNAL;
81      2      =      DCL (X,Y) ADDRESS,
          =      VECTOR (4) BYTE;
82      2      =      END;
          =
          =
          =
83      1      =      GRAPHIC$DESIG$2:PROCEDURE (X,Y,DESIG) EXTERNAL;
84      2      =      DCL (X,Y,DESIG) ADDRESS,
          =      VALUE BASED DESIG ADDRESS ,
          =      BUFFER (6) BYTE,
          =      (ROW,COLUMN) BYTE;
85      2      =      END;
          =
          =
          =
86      1      =      START$ERASE$VECTOR$2:PROCEDURE (X,Y) EXTERNAL;
87      2      =      DCL (X,Y) ADDRESS,

```



```

88      2      =      VECTOR (4) BYTE;
      2      =      END;
      2      =
      2      =
89      1      =      STOP$ERASE$VECTOR$2:PROCEDURE(X,Y) EXTERNAL;
90      2      =      DCL (X,Y) ADDRESS,
      2      =      VECTOR (4) BYTE;
91      2      =      END;
      2      =
      2      =
92      1      =      START$ERASE$DASH$2:PROCEDURE (X,Y) EXTERNAL;
93      2      =      DCL (X,Y) ADDRESS,
      2      =      VECTOR (4) BYTE;
94      2      =      END;
      2      =
      2      =
95      1      =      STOP$ERASE$DASH$2:PROCEDURE (X,Y) EXTERNAL;
96      2      =      DCL (X,Y) ADDRESS,
      2      =      VECTOR (4) BYTE;
97      2      =      END;
      2      =
      2      =

```



```

122      =      X=127;
123      =      Y=511;
124      =      CALL STOP$VECTOR$SOLID(X,Y);
125      =      X=255;
126      =      Y=0;
127      =      CALL START$VECTOR$SOLID(X,Y);
128      =      X=255;
129      =      Y=511;
130      =      CALL STOP$VECTOR$SOLID(X,Y);
131      =      X=383;
132      =      Y=0;
133      =      CALL START$VECTOR$SOLID(X,Y);
134      =      X=383;
135      =      Y=511;
136      =      CALL STOP$VECTOR$SOLID(X,Y);
137      =      X=0;
138      =      Y=127;
139      =      CALL START$VECTOR$SOLID(X,Y);
140      =      X=511;
141      =      Y=127;
142      =      CALL STOP$VECTOR$SOLID(X,Y);

```



```

143      X=0;
144      Y=255;
145      CALL START$VECTOR$SOLID(X,Y);
146      X=511;
147      Y=255;
148      CALL STOP$VECTOR$SOLID(X,Y);
149      X=0;
150      Y=383;
151      CALL START$VECTOR$SOLID(X,Y);
152      X=511;
153      Y=383;
154      CALL STOP$VECTOR$SOLID(X,Y);

/* LABEL GRID */
155      W=1;
156      Y=118;
157      DO I=1 TO 2;
158          X=4;
159          DO J = 1 TO 4;
160              CALL WRITE$CONTACT$ID(X,Y,W);
161              X=X+128;

```



```

162      W=W+1;
163      END; /* DO J */
164      Y=246;
165      END; /* DO I */
166      CALL WRITE$CONTACT$ID(4,374,9);
167      CALL WRITE$CONTACT$ID(4,502,1);
168      CALL WRITE$CONTACT$ID(10,502,3);
169      W=0;
170      Y=374;
171      DO I = 1 TO 2 ;
172      X=131;
173      DO J=1 TO 3 ;
174      CALL WRITE$CONTACT$ID(X,Y,1);
175      CALL WRITE$CONTACT$ID(X+6,Y,W);
176      X=X+128;
177      W=W+1;
178      END; /* DO J */
179      W=W+1;
180      Y=502;
181      END; /* DO I */
=

```



```

182 2 = END; /* DRAW$GRID */
    =
    =
    =
183 1 = DRAW$FRIEND$SYMBOL: PROCEDURE (X,Y,S) PUBLIC;
    =
    = /* THIS PROCEDURE DRAWS A SOLID CIRCLE WHICH REPRESENT
S
    =
    = A FRIENDLY SHIP */
184 2 = DCL (X,Y,TX,TY) ADDRESS;
185 2 = DCL S BYTE;
186 2 = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
    = THEN
187 2 = RETURN; /* CANNOT DRAW SYMBOL, NEAR EDGE OF SCREEN.
*/
188 2 = TX = S*(X-2);
189 2 = TY = S*(Y-6);
190 2 = CALL START$VECTOR$SOLID(TX,TY);
191 2 = TX = S*(X+2);
192 2 = TY = S*(Y-6);
193 2 = CALL STOP$VECTOR$SOLID(TX,TY);
194 2 = TX = S*(X+6);
195 2 = TY = S*(Y-2);

```



```

196      CALL STOP$VECTOR$SOLID(TX, TY);
197      TX = S*(X+6);
198      TY = S*(Y+2);
199      CALL STOP$VECTOR$SOLID(TX, TY);
200      TX = S*(X+2);
201      TY = S*(Y+6);
202      CALL STOP$VECTOR$SOLID(TX, TY);
203      TX = S*(X-2);
204      TY = S*(Y+6);
205      CALL STOP$VECTOR$SOLID(TX, TY);
206      TX = S*(X-6);
207      TY = S*(Y+2);
208      CALL STOP$VECTOR$SOLID(TX, TY);
209      TX = S*(X-6);
210      TY = S*(Y-2);
211      CALL STOP$VECTOR$SOLID(TX, TY);
212      TX = S*(X-2);
213      TY = S*(Y-6);
214      CALL STOP$VECTOR$SOLID(TX, TY);
215      END; /* DRAW$FRIEND$SYMBOL */
=

```



```

=
=
216 1      DRAW$FRIEND$DASH: PROCEDURE (X,Y,S) PUBLIC;
=
=      /* THIS PROCEDURE DRAWS A DASHED CIRCLE WHICH REPRESENTS
TS
217      A FRIENDLY SHIP */
218      DCL (X,Y,TX,TY) ADDRESS;
219      DCL S BYTE;
219      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
219      THEN
220      RETURN; /* CANNOT DRAW SYMBOL, NEAR EDGE OF SCREEN.
*/
221      TX = S*(X+6);
222      TY = S*(Y+2);
223      CALL START$VECTOR$DASH(TX,TY);
224      TX = S*(X+2);
225      TY = S*(Y+6);
226      CALL STOP$VECTOR$DASH(TX,TY);
227      TX = S*(X-2);
228      TY = S*(Y+6);
229      CALL START$VECTOR$DASH(TX,TY);
230      TX = S*(X-6);

```



```

231 2      TY = S*(Y+2);
232 2      CALL STOP$VECTOR$DASH(TX,TY);
233 2      END; /* DRAW$FRIEND$DASH */
     =
     =
234 1      ERASE$FRIEND$DASH: PROCEDURE (X,Y,S) PUBLIC;
     =
     =
     /* THIS PROCEDURE ERASES A DASHED CIRCLE WHICH REPRESE
     NTS
         A FRIENDLY SHIP */
235 2      DCL (X,Y,TX,TY) ADDRESS;
236 2      DCL S BYTE;
237 2      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
     =      THEN
238 2      RETURN; /* CANNOT ERASE SYMBOL, NEAR EDGE OF SCREEN.
     */
239 2      TX = S*(X+6);
240 2      TY = S*(Y+2);
241 2      CALL START$ERASE$DASH(TX,TY);
242 2      TX = S*(X+2);
243 2      TY = S*(Y+6);
244 2      CALL STOP$ERASE$DASH(TX,TY);

```



```
245 TX = S*(X-2);
246 TY = S*(Y+6);
247 CALL START$ERASE$DASH(TX, TY);
248 TX = S*(X-6);
249 TY = S*(Y+2);
250 CALL STOP$ERASE$DASH(TX, TY);
251 END; /* ERASE$FRIEND$DASH */
=
=
=
252 1 ERASE$FRIEND$SYMBOL: PROCEDURE (X, Y, S) PUBLIC;
=
=
=
TS /* THIS PROCEDURE ERASES A SOLID CIRCLE WHICH REPRESENTS
=
=
=
253 A FRIENDLY SHIP */
254 DCL (X, Y, TX, TY) ADDRESS;
255 DCL S BYTE;
=
=
=
256 IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
*/ THEN
257 RETURN; /* CANNOT ERASE SYMBOL, NEAR EDGE OF SCREEN.
258 TX = S*(X-2);
TY = S*(Y-6);
```



```
259 2 = CALL START$ERASE$VECTOR(TX, TY);
260 2 = TX = S*(X+2);
261 2 = TY = S*(Y-6);
262 2 = CALL STOP$ERASE$VECTOR(TX, TY);
263 2 = TX = S*(X+6);
264 2 = TY = S*(Y-2);
265 2 = CALL STOP$ERASE$VECTOR(TX, TY);
266 2 = TX = S*(X+6);
267 2 = TY = S*(Y+2);
268 2 = CALL STOP$ERASE$VECTOR(TX, TY);
269 2 = TX = S*(X+2);
270 2 = TY = S*(Y+6);
271 2 = CALL STOP$ERASE$VECTOR(TX, TY);
272 2 = TX = S*(X-2);
273 2 = TY = S*(Y+6);
274 2 = CALL STOP$ERASE$VECTOR(TX, TY);
275 2 = TX = S*(X-6);
276 2 = TY = S*(Y+2);
277 2 = CALL STOP$ERASE$VECTOR(TX, TY);
278 2 = TX = S*(X-6);
279 2 = TY = S*(Y-2);
```



```

280 2 = CALL STOP$ERASE$VECTOR(TX, TY);
281 2 = TX = S*(X-2);
282 2 = TY = S*(Y-6);
283 2 = CALL STOP$ERASE$VECTOR(TX, TY);
284 2 = END; /* ERASE$FRIEND$SYMBOL */
    =
    =
    =
    =
    =
285 1 = DRAW$HOSTILE$SYMBOL:PROCEDURE(X,Y,S) PUBLIC;
    =
    =
    =
    =
    =
286 2 = DCL (X,Y,TX,TY) ADDRESS;
287 2 = DCL S BYTE;
288 2 = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
    = THEN RETURN; /* CAN NOT DRAW SYMBOL */
290 2 = TX=S*(X);
291 2 = TY=S*(Y-6);

```



```

292 2 = CALL START$VECTOR$SOLID(TX, TY);
293 2 = TX=S*(X+6);
294 2 = TY=S*(Y);
295 2 = CALL STOP$VECTOR$SOLID(TX, TY);
296 2 = TX=S*(X);
297 2 = TY=S*(Y+6);
298 2 = CALL STOP$VECTOR$SOLID(TX, TY);
299 2 = TX=S*(X-6);
300 2 = TY=S*(Y);
301 2 = CALL STOP$VECTOR$SOLID(TX, TY);
302 2 = TX=S*(X);
303 2 = TY=S*(Y-6);
304 2 = CALL STOP$VECTOR$SOLID(TX, TY);
305 2 = END; /*DRAW$HOSTILE$SYMBOL */
=
=
=
=
306 1 = DRAW$HOSTILE$DASH:PROCEDURE(X, Y, S) PUBLIC;
=
= /* THIS PROCEDURE DRAWS A DASHED DIAMOND WHICH

```



```

=
=
307      REPRESENTS A HOSTILE SHIP. */
308      DCL (X,Y,TX,TY) ADDRESS;
309      DCL S BYTE;
      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      THEN RETURN; /* CAN NOT DRAW SYMBOL */
311      TX=S*(X);
312      TY=S*(Y+6);
313      CALL START$VECTOR$DASH(TX,TY);
314      TX=S*(X-6);
315      TY=S*(Y);
316      CALL STOP$VECTOR$DASH(TX,TY);
317      END; /*DRAW$HOSTILE$DASH */

=
=
=
=
318      ERASE$HOSTILE$SYMBOL:PROCEDURE(X,Y,S) PUBLIC;
      /* THIS PROCEDURE ERASES A SOLID DIAMOND WHICH
      REPRESENTS A HOSTILE SHIP. */
=
=
=
=
```



```

=
=
319 2      DCL (X,Y,TX,TY) ADDRESS;
320 2      DCL S BYTE;
321 2      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
=      THEN RETURN;      /* CAN NOT ERASE SYMBOL */
323 2      TX=S*(X);
324 2      TY=S*(Y-6);
325 2      CALL START$ERASE$VECTOR(TX,TY);
326 2      TX=S*(X+6);
327 2      TY=S*(Y);
328 2      CALL STOP$ERASE$VECTOR(TX,TY);
329 2      TX=S*(X);
330 2      TY=S*(Y+6);
331 2      CALL STOP$ERASE$VECTOR(TX,TY);
332 2      TX=S*(X-6);
333 2      TY=S*(Y);
334 2      CALL STOP$ERASE$VECTOR(TX,TY);
335 2      TX=S*(X);
336 2      TY=S*(Y-6);
337 2      CALL STOP$ERASE$VECTOR(TX,TY);
338 2      END; /*ERASE$HOSTILE$SYMBOL */
=

```



```

3339      1      ERASE$HOSTILE$DASH:PROCEDURE(X,Y,S) PUBLIC;
          1      /* THIS PROCEDURE ERASES A DASHED DIAMOND WHICH
          2      REPRESENTS A HOSTILE SHIP. */
          2      DCL (X,Y,TX,TY) ADDRESS;
          2      DCL S BYTE;
          2      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
          2      THEN RETURN; /* CAN NOT ERASE SYMBOL */
          2      TX=S*(X);
          2      TY=S*(Y+6);
          2      CALL START$ERASE$DASH(TX,TY);
          2      TX=S*(X-6);
          2      TY=S*(Y);
          2      CALL STOP$ERASE$DASH(TX,TY);
          2      END; /*ERASE$HOSTILE$DASH */
          2
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450

```



```
=
=
=
=
351 1 = DRAW$UNKNOWN$SYMBOL:PROCEDURE (X,Y,S) PUBLIC;
=
=
=
352 2 = /* THIS PROCEDURE DRAWS A SOLID SQUARE WHICH
353 2 = REPRESENTS AN UNKNOWN SHIP. */
354 2 = DCL (X,Y,TX,TY) ADDRESS;
=
=
=
356 2 = DCL S BYTE;
357 2 = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
358 2 = THEN RETURN; /* CAN NOT DRAW SYMBOL */
359 2 = TX=S*(X-5);
360 2 = TY=S*(Y-5);
361 2 = CALL START$VECTOR$SOLID(TX,TY);
362 2 = TX=S*(X+5);
363 2 = TY=S*(Y+5);
=
```



```
364      = CALL STOP$VECTOR$SOLID(TX, TY);
365      = TX=S*(X-5);
366      = TY=S*(Y+5);
367      = CALL STOP$VECTOR$SOLID(TX, TY);
368      = TX=S*(X-5);
369      = TY =S*(Y-5);
370      = CALL STOP$VECTOR$SOLID(TX, TY);
371      = END; /* DRAW$UNKNOWN$SYMBOL */

372      1 = ERASE$UNKNOWN$SYMBOL:PROCEDURE (X, Y, S) PUBLIC;
      =
      = /* THIS PROCEDURE ERASES A SOLID SQUARE WHICH
      = REPRESENTS AN UNKNOWN SHIP. */
      = DCL (X, Y, TX, TY) ADDRESS;
373      2 DCL S BYTE;
374      2 DCL S BYTE;
375      2 IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      = THEN RETURN; /* CAN NOT ERASE SYMBOL */
      = TX=S*(X-5);
377      2 TY=S*(Y-5);
378      2 CALL START$ERASE$VECTOR(TX, TY);
379      2
```


MAY 1

```

380      TX=S*(X+5);
381      TY=S*(Y-5);
382      CALL STOP$ERASE$VECTOR(TX,TY);
383      TX=S*(X+5);
384      TY=S*(Y+5);
385      CALL STOP$ERASE$VECTOR(TX,TY);
386      TX=S*(X-5);
387      TY=S*(Y+5);
388      CALL STOP$ERASE$VECTOR(TX,TY);
389      TX=S*(X-5);
390      TY =S*(Y-5);
391      CALL STOP$ERASE$VECTOR(TX,TY);
392      END; /* ERASE$UNKNOWN$SYMBOL */

393      1      DRAW$UNKNOWN$DASH:PROCEDURE (X,Y,S) PUBLIC;

          /* THIS PROCEDURE DRAWS A DASH SQUARE WHICH
             REPRESENTS AN UNKNOWN SHIP. */
394      2      DCL (X,Y,TX,TY) ADDRESS;
395      2      DCL S BYTE;

```



```

396      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      THEN RETURN; /* CAN NOT DRAW SYMBOL */
398      TX=S*(X+5);
399      TY=S*(Y+5);
400      CALL START$VECTOR$DASH(TX,TY);
401      TX=S*(X-5);
402      TY=S*(Y+5);
403      CALL STOP$VECTOR$DASH(TX,TY);
404      END; /* DRAW$UNKNOWN$DASH */

405      ERASE$UNKNOWN$DASH:PROCEDURE (X,Y,S) PUBLIC;

      /* THIS PROCEDURE ERASES A DASHED SQUARE WHICH
      REPRESENTS AN UNKNOWN SHIP. */
      DCL (X,Y,TX,TY) ADDRESS;
      DCL S BYTE;
      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      THEN RETURN; /* CAN NOT ERASE SYMBOL */
      TX=S*(X+5);
      TY=S*(Y+5);

```


MAY 1

```

412 = CALL START$ERASE$DASH(TX, TY);
413 = TX=S*(X-5);
414 = TY=S*(Y+5);
415 = CALL STOP$ERASE$DASH(TX, TY);
416 = END; /* ERASE$UNKNOWN$DASH */
      $INCLUDE(:F1:PLAPUB.TWO)
=
=
=

```

/* DECLARE PLASMA PUBLIC PROCEDURES FOR PLASMA DEVICE

TWO. */

```

417 = 1 WRITE$CONTACT$ID$2: PROCEDURE (X, Y, J) PUBLIC ;
=
=
=

```

/* THIS PROCEDURE WRITES CONTACTS ID ADJACENT TO SYMB

OLS. */

```

418 = 2 DCL (ROW, COLUMN, J) BYTE ;
419 = 2 DCL (X, Y) ADDRESS ;
420 = 2 DCL CONTACT$ID (3) BYTE ;
421 = 2 CONTACT$ID(0)=J+'0';
422 = 2 CONTACT$ID(1)='$';
423 = 2 CONTACT$ID(2)='$';
424 = 2 ROW=Y/16;
425 = 2 COLUMN=((X-14)/6);
=
=
=

```



```

426 IF (X<=8) THEN COLUMN=0;
428 IF (X>=9) AND (X<=44) THEN COLUMN=X/6;
430 IF (X>=494) THEN COLUMN=75;
432 CALL PLASMA$PRINT$STRING$2(COLUMN,ROW,.CONTACT$ID);
433 END; /* WRITE$CONTACT$ID$2 */
=
=
=
=
=
=
=
=
=
=
=
=
434 DRAW$GRID$2: PROCEDURE PUBLIC;
=
=
/* THIS PROCEDURE DRAWS A GRID IN PLASMA DEVICE ONE */
DCL (W,I,J) BYTE;
435 DCL (X,Y) ADDRESS;
436 CALL CLEAR$PLASMA$2;
437 X=127;
438 Y=0;
439 CALL START$VECTOR$SOLID$2(X,Y);
440 X=127;
441

```



```

442      =      Y=511;
443      2      CALL STOP$VECTOR$SOLID$2(X,Y);
444      =      X=255;
445      =      Y=0;
446      2      CALL START$VECTOR$SOLID$2(X,Y);
447      =      X=255;
448      2      Y=511;
449      =      CALL STOP$VECTOR$SOLID$2(X,Y);
450      2      X=383;
451      =      Y=0;
452      2      CALL START$VECTOR$SOLID$2(X,Y);
453      =      X=383;
454      2      Y=511;
455      =      CALL STOP$VECTOR$SOLID$2(X,Y);
456      2      X=0;
457      =      Y=127;
458      2      CALL START$VECTOR$SOLID$2(X,Y);
459      =      X=511;
460      2      Y=127;
461      =      CALL STOP$VECTOR$SOLID$2(X,Y);
462      2      X=0;

```



```

482      =      END; /* DO J */
483      =      Y=246;
484      =      END; /* DO I */
485      =      CALL WRITE$CONTACT$ID$2(4,374,9);
486      =      CALL WRITE$CONTACT$ID$2(4,502,1);
487      =      CALL WRITE$CONTACT$ID$2(10,502,3);
488      =      W=0;
489      =      Y=374;
490      =      DO I = 1 TO 2 ;
491      =      X=131;
492      =      DO J = 1 TO 3 ;
493      =      CALL WRITE$CONTACT$ID$2(X,Y,1);
494      =      CALL WRITE$CONTACT$ID$2(X+6,Y,W);
495      =      X=X+128;
496      =      W=W+1;
497      =      END; /* DO J */
498      =      W=W+1;
499      =      Y=502;
500      =      END; /* DO I */
501      =      END; /* DRAW$GRID$2 */

```



```

=
=
=
502 1      DRAW$FRIEND$SYMBOL$2: PROCEDURE (X,Y,S) PUBLIC;
=
=      /* THIS PROCEDURE DRAWS A SOLID CIRCLE WHICH REPRESENT
S      A FRIENDLY SHIP */
503      DCL (X,Y,TX,TY) ADDRESS;
504      DCL S BYTE;
505      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
=      THEN
=      RETURN; /* CANNOT DRAW SYMBOL, NEAR EDGE OF SCREEN.
*/
507      TX = S*(X-2);
508      TY = S*(Y-6);
509      CALL START$VECTOR$SOLID$2(TX,TY);
510      TX = S*(X+2);
511      TY = S*(Y-6);
512      CALL STOP$VECTOR$SOLID$2(TX,TY);
513      TX = S*(X+6);
514      TY = S*(Y-2);
515      CALL STOP$VECTOR$SOLID$2(TX,TY);
=

```


MAY 1

```

535 1 = DRAW$FRIEND$DASH$2: PROCEDURE (X,Y,S) PUBLIC;
    =
    =
    =
    TS
    =
536 2 = A FRIENDLY SHIP */
537 2 = DCL (X,Y,TX,TY) ADDRESS;
538 2 = DCL S BYTE;
    = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
    = THEN
539 2 = RETURN; /* CANNOT DRAW SYMBOL, NEAR EDGE OF SCREEN.
    */
540 2 = TX = S*(X+6);
541 2 = TY = S*(Y+2);
542 2 = CALL START$VECTOR$DASH$2(TX,TY);
543 2 = TX = S*(X+2);
544 2 = TY = S*(Y+6);
545 2 = CALL STOP$VECTOR$DASH$2(TX,TY);
546 2 = TX = S*(X-2);
547 2 = TY = S*(Y+6);
548 2 = CALL START$VECTOR$DASH$2(TX,TY);
549 2 = TX = S*(X-6);
550 2 = TY = S*(Y+2);

```



```

565      TY = S*(Y+6);
566      CALL START$ERASE$DASH$2(TX,TY);
567      TX = S*(X-6);
568      TY = S*(Y+2);
569      CALL STOP$ERASE$DASH$2(TX,TY);
570      END; /* ERASE$FRIEND$DASH$2 */
      =
      =
      =
571      ERASE$FRIEND$SYMBOL$2: PROCEDURE (X,Y,S) PUBLIC;
      =
      =
      =
      /* THIS PROCEDURE ERASES A SOLID CIRCLE WHICH REPRESENTS
      A FRIENDLY SHIP */
572      DCL (X,Y,TX,TY) ADDRESS;
573      DCL S BYTE;
574      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      THEN
575      RETURN; /* CANNOT ERASE SYMBOL, NEAR EDGE OF SCREEN.
      */
576      TX = S*(X-2);
577      TY = S*(Y-6);
578      CALL START$ERASE$VECTOR$2(TX,TY);

```



```

579      TX = S*(X+2);
580      TY = S*(Y-6);
581      CALL STOP$ERASE$VECTOR$2(TX, TY);
582      TX = S*(X+6);
583      TY = S*(Y-2);
584      CALL STOP$ERASE$VECTOR$2(TX, TY);
585      TX = S*(X+6);
586      TY = S*(Y+2);
587      CALL STOP$ERASE$VECTOR$2(TX, TY);
588      TX = S*(X+2);
589      TY = S*(Y+6);
590      CALL STOP$ERASE$VECTOR$2(TX, TY);
591      TX = S*(X-2);
592      TY = S*(Y+6);
593      CALL STOP$ERASE$VECTOR$2(TX, TY);
594      TX = S*(X-6);
595      TY = S*(Y+2);
596      CALL STOP$ERASE$VECTOR$2(TX, TY);
597      TX = S*(X-6);
598      TY = S*(Y-2);
599      CALL STOP$ERASE$VECTOR$2(TX, TY);

```



```
600 2 = TX = S*(X-2);
601 2 = TY = S*(Y-6);
602 2 = CALL STOP$ERASE$VECTOR$2(TX,TY);
603 2 = END; /* ERASE$FRIEND$SYMBOL$2 */
    =
    =
    =
    =
604 1 = DRAW$HOSTILE$SYMBOL$2:PROCEDURE(X,Y,S) PUBLIC;
    =
    =
    =
    =
    =
605 2 = DCL (X,Y,TX,TY) ADDRESS;
606 2 = DCL S BYTE;
607 2 = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
    = THEN RETURN; /* CAN NOT DRAW SYMBOL */
609 2 = TX=S*(X);
610 2 = TY=S*(Y-6);
611 2 = CALL START$VECTOR$SOLID$2(TX,TY);
```



```

1 658 ERASE$HOSTILE$DASH$2:PROCEDURE(X,Y,S) PUBLIC;
/* THIS PROCEDURE ERASES A DASHED DIAMOND WHICH
REPRESENTS A HOSTILE SHIP. */
DCL (X,Y,TX,TY) ADDRESS;
DCL S BYTE;
IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
THEN RETURN; /* CAN NOT ERASE SYMBOL */
TX=S*(X);
TY=S*(Y+6);
CALL START$ERASE$DASH$2(TX,TY);
TX=S*(X-6);
TY=S*(Y);
CALL STOP$ERASE$DASH$2(TX,TY);
END; /*ERASE$HOSTILE$DASH$2 */
2 659
2 660
2 661
2 663
2 664
2 665
2 666
2 667
2 668
2 669

```



```

=
=
=
=
670 1      /* END PLASMA PUBLICS FOR PLASMA DEVICE TWO. */
      $INCLUDE(:F1:UNKNOWN.SR2)
=
=
671 2      DRAW$UNKNOWN$SYMBOL$2:PROCEDURE (X,Y,S) PUBLIC;
672 2
673 2      /* THIS PROCEDURE DRAWS A SOLID SQUARE WHICH
        DCL (X,Y,TX,TY) ADDRESS;
        DCL S BYTE;
        IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
        THEN RETURN; /* CAN NOT DRAW SYMBOL */
        TX=S*(X-5);
        TY=S*(Y-5);
        CALL START$VECTOR$SOLID$2(TX,TY);
        TX=S*(X+5);
        TY=S*(Y+5);
        CALL STOP$VECTOR$SOLID$2(TX,TY);
        TX=S*(X+5);
        TY=S*(Y+5);
675 2
676 2
677 2
678 2
679 2
680 2
681 2
682 2
```



```

683      = CALL STOP$VECTOR$SOLID$2(TX, TY);
684      = TX=S*(X-5);
685      = TY=S*(Y+5);
686      = CALL STOP$VECTOR$SOLID$2(TX, TY);
687      = TX=S*(X-5);
688      = TY =S*(Y-5);
689      = CALL STOP$VECTOR$SOLID$2(TX, TY);
690      = END; /* DRAW$UNKNOWN$SYMBOL$2 */

691      = ERASE$UNKNOWN$SYMBOL$2:PROCEDURE (X,Y,S) PUBLIC;

      = /* THIS PROCEDURE ERASES A SOLID SQUARE WHICH
      = REPRESENTS AN UNKNOWN SHIP. */
      = DCL (X,Y,TX,TY) ADDRESS;
      = DCL S BYTE;
      = IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      = THEN RETURN; /* CAN NOT ERASE SYMBOL */
      = TX=S*(X-5);
      = TY=S*(Y-5);
      = CALL START$ERASE$VECTOR$2(TX, TY);

```



```

699      TX=S*(X+5);
700      TY=S*(Y-5);
701      CALL STOP$ERASE$VECTOR$2(TX,TY);
702      TX=S*(X+5);
703      TY=S*(Y+5);
704      CALL STOP$ERASE$VECTOR$2(TX,TY);
705      TX=S*(X-5);
706      TY=S*(Y+5);
707      CALL STOP$ERASE$VECTOR$2(TX,TY);
708      TX=S*(X-5);
709      TY =S*(Y-5);
710      CALL STOP$ERASE$VECTOR$2(TX,TY);
711      END; /* ERASE$UNKNOWN$SYMBOL$2 */

712      DRAW$UNKNOWN$DASH$2:PROCEDURE (X,Y,S) PUBLIC;

      /* THIS PROCEDURE DRAWS A DASH SQUARE WHICH
      REPRESENTS AN UNKNOWN SHIP. */
713      DCL (X,Y,TX,TY) ADDRESS;
714      DCL S BYTE;

```



```

715 2      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      = THEN RETURN; /* CAN NOT DRAW SYMBOL */
717 2      TX=S*(X+5);
718 2      TY=S*(Y+5);
719 2      CALL START$VECTOR$DASH$2(TX,TY);
720 2      TX=S*(X-5);
721 2      TY=S*(Y+5);
722 2      CALL STOP$VECTOR$DASH$2(TX,TY);
723 2      END; /* DRAW$UNKNOWN$DASH$2 */
      =
      =
724 1      ERASE$UNKNOWN$DASH$2:PROCEDURE (X,Y,S) PUBLIC;
      =
      =
      = /* THIS PROCEDURE ERASES A DASHED SQUARE WHICH
      = REPRESENTS AN UNKNOWN SHIP. */
725 2      DCL (X,Y,TX,TY) ADDRESS;
726 2      DCL S BYTE;
727 2      IF (X<7) OR (X>504) OR (Y<7) OR (Y>504)
      = THEN RETURN; /* CAN NOT ERASE SYMBOL */
729 2      TX=S*(X+5);
730 2      TY=S*(Y+5);

```



```

731 2 = CALL START$ERASE$DASH$2(TX, TY);
732 2 = TX=S*(X-5);
733 2 = TY=S*(Y+5);
734 2 = CALL STOP$ERASE$DASH$2(TX, TY);
735 2 = END; /* ERASE$UNKNOWN$DASH$2 */

736 1      END; /* PLASMA$MODULE */

```

MODULE INFORMATION:

```

CODE AREA SIZE      = 1E5EH      7774D
VARIABLE AREA SIZE = 014EH      334D
MAXIMUM STACK SIZE = 0006H      6D
1072 LINES READ
0 PROGRAM ERROR(S)

```

END OF PL/M-80 COMPILATION

ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE PLASMAPRIMITIVES
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY: PLM80 :F1:PSPRIM.SRC NOOBJECT PAGEWIDTH(92) PA
GELENGTH(24) D
-ATE(MAY 1979) TITLE('MODULE PSPRIM.SRC')

1

PLASMA\$PRIMITIVES: DO;

/*****

*
*
*
*
*
*
*
*
*
*

THIS MODULE CONTAINS THE PRIMITIVES FOR
PLASMA DEVICE ONE AND PLASMA DEVICE TWO.

MDS I/O PORT ASSIGNMENTS:
PLASMA DEVICE ONE: PORTS 4 AND 5.
PLASMA DEVICE TWO: PORTS 8 AND 9.

*****/


```

2 1 DECLARE LIT LITERALLY 'LITERALLY',
   DCL LIT 'DECLARE';

3 1 DCL TRUE LIT '0FFH';
   FALSE LIT '00H';

4 1 DCL PLASMA$DATA$ONE LIT '04H', /* PLASMA DEVICE
   ONE */
   PLASMA$STATUS$ONE LIT '05H', /* PLASMA DEVICE
   ONE */
   PLASMA$DATA$TWO LIT '08H', /* PLASMA DEVICE T
   WO */
   PLASMA$STATUS$TWO LIT '09H', /* PLASMA DEVICE T
   WO */
   RECEIVE$MASK LIT '06H',

```



```
5      1      DCL STATUS$A      TRANSMIT$MASK      LIT '05H';
          RESET$ALL
          OUT$BUSY
          IN$BUSY

6      1      DCL STX           /* START TEXT */
          FTX                   /* ENABLE TEXT */
          CS                    /* CLEAR SCREEN */
          CG                    /* CONSTRUCT GRAPH */

          CV                    /* CLEAR VECTORS */
          EOL                   /* END OF LINE */

7      1      DCL SET$ERASE      LIT '0100$0000B';
          SET$DASHED            LIT '0001$0000B';
          SET$END                LIT '0010$0000B';
```



```

8 1  SET$STATUS$PLASMA: PROCEDURE (STATUS) PUBLIC;
9 2  DCL STATUS BYTE;
10 2  OUTPUT(PLASMA$STATUS$ONE) = NOT STATUS;
11 2  END SET$STATUS$PLASMA;

```

/*****

*

* PLASMA\$WRITE:

* THIS PROCEDURE IS USED TO SEND A CHARACTER TO THE

PLASMA DISPLAY

*

* PARAMETERS:

* - CHAR. - ASCII CHARACTER DESIRED TO BE SENT.

*

```

12 1  PLASMA$WRITE: PROCEDURE (CHAR) PUBLIC;

```



```
13 2      DCL CHAR BYTE;  
14 2      DO WHILE ((NOT INPUT(PLASMA$STATUS$ONE)) AND STATUS  
$A) <> STATUS$  
-  
15 3      END;  
16 2      /* WAIT FOR PLASMA TO BE READY */  
17 2      CALL SET$STATUS$PLASMA(RESET$ALL);  
18 2      OUTPUT(PLASMA$DATA$ONE) = NOT CHAR;  
19 2      CALL SET$STATUS$PLASMA(OUT$BUSY);  
      END PLASMA$WRITE;
```

```
*****  
-  
*****  
*  
* CLEAR$PLASMA:  
* THIS PROCEDURE IS USED TO CLEAR THE PLASMA DISPLAY  
*  
*****  
*****  
*****  
-  
*****/  
20 1      CLEAR$PLASMA: PROCEDURE PUBLIC;  
21 2      CALL SET$STATUS$PLASMA(IN$BUSY);
```



```

222 CALL SET$STATUS$PLASMA(RESET$ALL);
223 CALL PLASMA$WRITE(CS);
224 CALL PLASMA$WRITE(CV);
225 CALL SET$STATUS$PLASMA(RESET$ALL);
226 END CLEAR$PLASMA;

```

✱✱✱✱✱✱

✻

**** PLASMA\$WRITE\$VECTOR:**

*** THIS PROCEDURE IS USED TO SEND A FOUR BYTE VECTOR**

TO THE PLASMA.

**** PARAMETERS:**

* - A. - POINTER TO THE FOUR BYTE VECTOR DESIRED TO B
E SENT.

```

1 1 PLASMA$WRITE$VECTOR: PROCEDURE (A) PUBLIC;
2 2 DCL A ADDRESS.

```



```

29      VECTOR BASED A (4) BYTE,
30      VPTR BYTE;
31      DO VPTR = 0 TO 3;
32      CALL PLASMA$WRITE(VECTOR(VPTR));
      END;
      END PLASMA$WRITE$VECTOR;

```

/*****

- *****

*
* PLASMA\$PRINT\$STRING:

* THIS PROCEDURE IS USED TO WRITE A GIVEN STRING IN

A GIVEN POSITI

- ON AT

* THE PLASMA DISPLAY.

*
* PARAMETERS:

* - COLUMN.- DENOTES THE COLUMN NUMBER DESIRED TO BE

ADDRESSED.

* - ROW.- DENOTES THE ROW NUMBER DESIRED TO BE ADDRE

SSSED.

* - POINTER.- POINTS TO THE FIRST BYTE OF THE STRING

DESIRED TO BE


```

-      *      DIS-
MARK THE END      *      PLAYED. NOTE THAT TWO CONSECUTIVE '^' SIGNS MUST
-      *      OF
-      *      THE STRING.
-      *
*****
*****/
33 1 PLASMA$PRINT$STRING: PROCEDURE (COLUMN, ROW, POINTER)
PUBLIC;
34 2
35 2 COUNT = 0;
36 2 CALL SET$STATUS$PLASMA(IN$BUSY);
37 2 CALL SET$STATUS$PLASMA(RESET$ALL);
38 2 CALL PLASMA$WRITE(STX);
39 2 CALL PLASMA$WRITE(COLUMN);
40 2 CALL PLASMA$WRITE(ROW);
41 2 DO WHILE (BUFFER(COUNT) <> EOL ) OR (BUFFER(COUNT) +
1) <> EOL );
42 3 CALL PLASMA$WRITE(BUFFER(COUNT));
43 3 COUNT = COUNT + 1;
44 3 END;
```



```

56      2      VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
57      2      VECTOR(1) = LOW(X) AND 07FH;
58      2      VECTOR(2) = LOW(Y) AND 07FH;
59      2      VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
60              HIGH(SHL(X AND 180H, 1));
61      2      VECTOR(3) = VECTOR(3) AND NOT SET$ERASE;
            2      END SET$VECTOR;

```

```

62      1      SET$VECTOR$1: PROCEDURE (X, Y, POINTER);
63      2      DCL (X, Y, POINTER) ADDRESS,
            2      VECTOR BASED POINTER (4) BYTE;
64      2      VECTOR(0) = CG;
65      2      VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
66      2      VECTOR(1) = LOW(X) AND 07FH;
67      2      VECTOR(2) = LOW(Y) AND 07FH;
68      2      VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
            2      HIGH(SHL(X AND 180H, 1));
69      2      END SET$VECTOR$1;

```



```

72      2      CALL SET$VECTOR(X, Y, .VECTOR);
73      2      VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
74      2      VECTOR(3) = VECTOR(3) AND NOT SET$END;
75      2      CALL PLASMA$WRITE$VECTOR(.VECTOR);
76      2      END START$VECTOR$SOLID;

```

```

/*****
****

```

*

STOP\$VECTOR\$SOLID:

THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR

A SOLID VECTOR

*

PARAMETERS:

- X.- ADDRESS VALUE.
- Y.- ADDRESS VALUE.

```

****
****

```

****/

-


```

77 1      STOP$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
78 2      DCL (X, Y) ADDRESS,
79 3      VECTOR (4) BYTE;
80 4      CALL SET$VECTOP(X, Y, .VECTOR);
81 5      VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
82 6      VECTOR(3) = VECTOR(3) OR SET$END;
83 7      CALL PLASMA$WRITE$VECTOR(.VECTOR);
      END STOP$VECTOR$SOLID;

```

```

*****/*****
-      *****
*      *****
*      START$VECTOR$DASH:
*      THIS PROCEDURE IS USED TO DEFINE A START POINT FOR
A DASHED VECT
-      OR.
*
* PARAMETERS:
* - X.- ADDRESS VALUE.
* - Y.- ADDRESS VALUE.

```



```

*
*****
*****/
84 1 START$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
85 2 DCL (X, Y) ADDRESS,
      VECTOR (4) BYTE;
86 2 CALL SET$VECTOR(X, Y, .VECTOR);
87 2 VECTOR(3) = VECTOR(3) OR SET$DASHED;
88 2 VECTOR(3) = VECTOR(3) AND NOT SET$END;
89 2 CALL PLASMA$WRITE$VECTOR(.VECTOR);
90 2 END START$VECTOR$DASH;

*****
*****/
*****
*****
*
* STOP$VECTOR$DASH:
* THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR
A DASHED VECTO
- R.
*
```



```

105      2      ROW = Y / 16;
106      2      COLUMN = ((X - 14) / 6) - 5;
107      2      IF X <= 8 THEN COLUMN = 0;
109      2      IF (X >= 9) AND (X <= 44) THEN COLUMN = X / 6;
111      2      IF (X >= 494) THEN COLUMN = 75;
113      2      CALL PLASMA$PRINT$STRING(COLUMN, ROW, .BUFFER);
114      2      END GRAPHIC$DESIG;

```

/*****

*
*
*

START\$ERASE\$VECTOR :

*****/

```

115      1      START$ERASE$VECTOR : PROCEDURE (X,Y) PUBLIC ;
116      2      DCL (X,Y) ADDRESS ,
117      2      VECTOR(4) BYTE;
118      2      CALL SET$VECTOR$1 (X,Y, .VECTOR);
118      2      VECTOR(3) = VECTOR(3) OR SET$ERASE;

```



```

119      VECTOR(3)=VECTOR(3) AND NOT SET$DASHED;
120      VECTOR(3)=VECTOR(3) AND NOT SET$END;
121      CALL PLASMA$WRITE$VECTOR(.VECTOR);
122      END START$ERASE$VECTOR;

```

/*****

```

*
*      STOP$ERASE$VECTOR :
*
*****

```

/*****

*****/

```

123      1      STOP$ERASE$VECTOR : PROCEDURE (X,Y) PUBLIC ;
124      2      DCL (X,Y) ADDRESS ,
125      2      VECTOR (4) BYTE ;
126      2      CALL SET$VECTOR$1 (X,Y,.VECTOR) ;
127      2      VECTOR(3) = VECTOR(3) OR SET$ERASE;
128      2      VECTOR(3)=VECTOR(3) AND NOT SET$DASHED ;
129      2      VECTOR(3)=VECTOR(3) OR SET$END ;
130      2      CALL PLASMA$WRITE$VECTOR(.VECTOR) ;
          END STOP$ERASE$VECTOR ;

```



```

131 *****
132 /*****
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



```

*****
/*****
*
*      STOP$ERASE$DASH:
*
*****
*****/

139 1  STOP$ERASE$DASH: PROCEDURE (X,Y) PUBLIC;
140 2  DCL (X,Y) ADDRESS,
141 2  VECTOR (4) BYTE;
142 2  CALL SET$VECTOR$1 (X,Y , .VECTOR);
143 2  VECTOR(3) = VECTOR(3) OR SET$ERASE;
144 2  VECTOR(3) = VECTOR(3) OR SET$DASHED;
145 2  VECTOR(3) = VECTOR(3) OR SET$END;
146 2  CALL PLASMA$WRITE$VECTOR( .VECTOR);
      END STOP$ERASE$DASH;

/**** END PLASMA$PRIMITIVES$1: ****/

```



```

$INCLUDE(:F1:PSPR2.SRC)
/*** PLASMA$PRIMITIVES$2: ***/

```

```

=
=
=
=
=
=
=

```

```

/*****

```

```

*****

```

```

-

```

```

*

```

```

* SET$STATUS$PLASMA:

```

```

* THIS PROCEDURE IS USED TO SET THE STATUS LINE FOR

```

```

THE PLASMA.

```

```

* NOTE THAT THE LOGIC TO BE USED IS NEGATIVE.

```

```

*

```

```

* PARAMETERS:

```

```

* - STATUS.- ASCII CHARACTER USED TO DEFINE THE STAT

```

```

US LINE.

```



```

-      ****/
151 1 = PLASMA$WRITE$2: PROCEDURE (CHAR) PUBLIC;
152 2 =   DCL CHAR BYTE;
153 2 =   DO WHILE ((NOT INPUT(PLASMA$STATUS$TWO)) AND STATUS
$A) <> STATUS$
-      A;
154 3 =   END;          /* WAIT FOR PLASMA TO BE READY */
155 2 =   CALL SET$STATUS$PLASMA$2(RESET$ALL);
156 2 =   OUTPUT(PLASMA$DATA$TWO) = NOT CHAR;
157 2 =   CALL SET$STATUS$PLASMA$2(OUT$BUSY);
158 2 =   END PLASMA$WRITE$2;
=
=
=
=
*****/
-      ****
=
=
=
* CLEAR$PLASMA:
* THIS PROCEDURE IS USED TO CLEAR THE PLASMA DISPLAY
=
=
*
*****/
-      ****/

```



```

166 1 = PLASMA$WRITE$VECTOR$2: PROCEDURE (A) PUBLIC;
167 2 =   DCL A ADDRESS,
      3 =   VECTOR BASED A (4) BYTE,
      4 =   VPTR BYTE;
168 2 =   DO VPTR = 0 TO 3;
169 3 =     CALL PLASMA$WRITE$2(VECTOR(VPTR));
170 3 =   END;
171 2 =   END PLASMA$WRITE$VECTOR$2;

```

/*****

* PLASMA\$PRINT\$STRING:

* THIS PROCEDURE IS USED TO WRITE A GIVEN STRING IN

A GIVEN POSITI

ON AT

* THE PLASMA DISPLAY.

* PARAMETERS:

* - COLUMN.- DENOTES THE COLUMN NUMBER DESIRED TO BE

ADDRESSED.


```

= * - ROW.- DENOTES THE ROW NUMBER DESIRED TO BE ADDRE
SSSED.
= * - POINTER.- POINTS TO THE FIRST BYTE OF THE STRING
DESIRED TO BE
- DIS-
= * PLAYED. NOTE THAT TWO CONSECUTIVE '$' SIGNS MUST
MARK THE END
- OF
= * THE STRING.
= *
= *****
*****
*****/
172 1 = PLASMA$PRINT$STRING$2: PROCEDURE (COLUMN, ROW, POINTER
) PUBLIC;
173 2 =
= DCL POINTER ADDRESS,
= BUFFER BASED POINTER (1) BYTE,
= (COLUMN, ROW, COUNT) BYTE;
= COUNT = 0;
174 2 = CALL SET$STATUS$PLASMA$2(IN$BUSY);
175 2 = CALL SET$STATUS$PLASMA$2(RESET$ALL);
176 2 = CALL PLASMA$WRITE$2(STX);
177 2 = CALL PLASMA$WRITE$2(COLUMN);
178 2 = CALL PLASMA$WRITE$2(ROW);
179 2 = DO WHILE (BUFFER(COUNT) <> EOL ) OR (BUFFER(COUNT +
180 2 = 1) <> EOL );
1) <> EOL );
181 3 = CALL PLASMA$WRITE$2(BUFFER(COUNT));

```


MAY 1

```

194      VECTOR(0) = CG;
195      VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
196      VECTOR(1) = LOW(X) AND 07FH;
197      VECTOR(2) = LOW(Y) AND 07FH;
198      VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
199                  HIGH(SHL(X AND 180H, 1));
200      VECTOR(3) = VECTOR(3) AND NOT SET$ERASE;
      END SET$VECTOR$2;

```

```

201      SET$VECTOR$1$2: PROCEDURE (X, Y, POINTER);
202      DCL (X, Y, POINTER) ADDRESS,
203          VECTOR BASED POINTER (4) BYTE;
204      VECTOR(0) = CG;
205      VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
206      VECTOR(1) = LOW(X) AND 07FH;
207      VECTOR(2) = LOW(Y) AND 07FH;
208      VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR

```



```

208      2      =      HIGH(SHL(X AND 180H, 1));
      END SET$VECTOR$1$2;

```

```

/*****

```

```

*****

```

```

-

```

```

*

```

```

* START$VECTOR$SOLID:

```

```

* THIS PROCEDURE IS USED TO DEFINE A START POINT FOR

```

```

A SOLID VECTO

```

```

-

```

```

R.

```

```

*

```

```

* PARAMETERS:

```

```

* - X.- ADDRESS VALUE.

```

```

* - Y.- ADDRESS VALUE.

```

```

*

```

```

*****

```

```

*****

```

```

-

```

```

209      1      =      START$VECTOR$SOLID$2: PROCEDURE (X, Y) PUBLIC;

```


MAY 1

```

*****
= *****
- *****/
216 1 = STOP$VECTOR$SOLID$2: PROCEDURE (X, Y) PUBLIC;
217 2 = DCL (X, Y) ADDRESS,
      VECTOR (4) BYTE;
218 2 = CALL SET$VECTOR$2(X, Y, .VECTOR);
219 2 = VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
220 2 = VECTOR(3) = VECTOR(3) OR SET$END;
221 2 = CALL PLASMA$WRITE$VECTOR$2(.VECTOR);
222 2 = END STOP$VECTOR$SOLID$2;
=
=
=
=
/*****
*****
- *****
=
=
=
* START$VECTOR$DASH:
* THIS PROCEDURE IS USED TO DEFINE A START POINT FOR
A DASHED VECT
- OR.
=
=
* PARAMETERS:

```


MAY 1

```

*
* GRAPHIC$DESIG:
* THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATION
OF A CONTACT I - N THE
VALUES GIVEN. * NEAREST POSSIBLE ALPHANUMERIC LOCATION TO THE X, Y
*
* PARAMETERS:
* - X.- ADDRESS VALUE.
* - Y.- ADDRESS VALUE.
* - DESIG.- POINTER TO THE DESIG OF A CONTACT.
*****
*****/
237 1 = GRAPHIC$DESIG$2: PROCEDURE (X, Y, DESIG) PUBLIC;
238 2 = DCL (X, Y, DESIG) ADDRESS,
VALUE BASED DESIG ADDRESS,
BUFFER (6) BYTE,
(Row, Column) BYTE;
239 2 = BUFFER(0) = '[';
240 2 = BUFFER(1) = VALUE / 100;
241 2 = BUFFER(2) = VALUE MOD 100;

```



```

254 1 = START$ERASE$VECTOR$2: PROCEDURE (X,Y) PUBLIC ;
255 2 = DCL (X,Y) ADDRESS ,
      = VECTOR(4) BYTE;

```



```

256      = CALL SET$VECTOR$1$2 (X,Y, .VECTOR);
257      = VECTOR(3) = VECTOR(3) OR SET$ERASE;
258      = VECTOR(3)=VECTOR(3) AND NOT SET$DASHED;
259      = VECTOR(3)=VECTOR(3) AND NOT SET$END;
260      = CALL PLASMA$WRITE$VECTOR$2(.VECTOR);
261      = END START$ERASE$VECTOR$2;

```

/***** **** */

*
*
*
*

STOP\$ERASE\$VECTOR :

***** **** */

*****/

```

262      = STOP$ERASE$VECTOR$2: PROCEDURE (X,Y) PUBLIC ;
263      =   DCL (X,Y) ADDRESS ,
264      =   VECTOR (4) BYTE ;
265      =   CALL SET$VECTOR$1$2 (X,Y, .VECTOR) ;
266      =   VECTOR(3) = VECTOR(3) OR SET$ERASE;
267      =   VECTOR(3)=VECTOR(3) AND NOT SET$DASHED ;
267      =   VECTOR(3)=VECTOR(3) OR SET$END ;

```



```
268      = CALL PLASMA$WRITE$VECTOR$2(.VECTOR) ;
269      = END STOP$ERASE$VECTOR$2;
```

```
/*****
```

```
*****
```

```
      * START$ERASE$DASH:
      *
      *
```

```
*****
```

```
*****/
```

```
270      = START$ERASE$DASH$2: PROCEDURE (X,Y) PUBLIC;
271      =   DCL (X,Y) ADDRESS,
272      =   VECTOR (4) BYTE;
273      =   CALL SET$VECTOR$1$2 (X,Y,.VECTOR);
274      =   VECTOR(3) = VECTOR(3) OR SET$ERASE;
275      =   VECTOR(3) = VECTOR(3) OR SET$DASHED;
276      =   VECTOR(3) = VECTOR(3) AND NOT SET$END;
277      =   CALL PLASMA$WRITE$VECTOR$2(.VECTOR);
      =   END START$ERASE$DASH$2;
```



```

= = =
*****
/*****
*
*
*
*
*****
STOP$ERASE$DASH:
*****
*****/
278 1 STOP$ERASE$DASH$2: PROCEDURE (X,Y) PUBLIC;
279 2 DCL (X,Y) ADDRESS,
280 2 VECTOR (4) BYTE;
281 2 CALL SET$VECTOR$1$2 (X,Y,.VECTOR);
282 2 VECTOR(3) = VECTOR(3) OR SET$ERASE;
283 2 VECTOR(3) = VECTOR(3) OR SET$DASHED;
284 2 VECTOR(3) = VECTOR(3) OR SET$END;
285 2 CALL PLASMA$WRITE$VECTOR$2(.VECTOR);
END STOP$ERASE$DASH$2;
= = =

```


= /*** END PLASMA\$PRIMITIVES\$2 *** /

286 1 END PLASMA\$PRIMITIVES;

MODULE INFORMATION:

CODE AREA SIZE = 082CH 2092D
VARIABLE AREA SIZE = 00C8H 200D
MAXIMUM STACK SIZE = 0006H 6D
789 LINES READ
0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION


```

struct {
    int lat(10) ;
    int longtd(10) ;
    int course (10) ;
    char speed(10) ;
    int xbow(10) ;
    int ybow(10) ;
    char quadrant(10) ;
    int range(10) ;
    int bearing(10) ;
    char collflag(10) ;
    int cpatime(10) ;
    int cpadist(10) ;
    char count;
} shipplot(3), *shipptr;

```

```

struct {
    int xtail(10) ;
    int ytail(10) ;
    int rangetail(10);
    int heartail(10);
} back(3) ;

```

```

abs() ;
double fabs() ;
double atan() ;

```



```

double pow() ;
double sqrt() ;
double sin() ;
double cos() ;

float baselat,baselonq,latoff,lonqoff ;
int  fx1,fy1,fx2,fy2,fx0,fy0;
int  hx1,hy1,hx2,hy2,hx3,hy3,hx0,hy0;
int  ux1,uy1,ux0,uy0;
int  fsp1,fsp2,fsp0,hsp1,hsp2,hsp3,usp1,usp0;

int  sp10 10;
int  sp20 20;
int  sp30 30;
int  sp40 40;
int  intct; /* structure initialization counter */
int  times; /* sets counter increment */

char backship 0;
char cpaact;

char c,d,e,r,z;
int  k,e0,f0;

char cr 015;
char lf 012;
int  bitx7 000200;
int  bitx14 040000;

```



```

main()
{
    sleep(120);
    init0();
    cpact = 0;
    baselat = 40.0 ;
    baselong = 40.0 ;
    latoff = 0.05 ;
    longoff = 0.05 ;
    intct = 0;
    times = 1;
    /* zero xtail and ytail */
    for (c=0;c<3;c++)
    {
        for (d=0;d<10;d++)
        {
            backcl.xtail[d] = 1000;
            backcl.ytail[d] = 1000;
            backcl.ranqtail[d]=1000;
            backcl.beartail[d]=1000;
        }
    }
    /* initialize structure positions and speed */
    fx0=431;
    fy0=411;
    fx1=401;
    fy1=411;
    fx2=401;
    fy2=411;

```



```

hx0=192;
hy0=258;
hx1=168;
hy1=288;
hx2=168;
hy2=308;
hx3=192;
hy3=308;
ux0=40;
uy1=248;
ux1=490;
uy0=265;
fsp0=sp20;
fsp1=sp20;
fsp2=sp20;
hsp0=sp10;
hsp1=sp10;
hsp2=sp10;
hsp3=sp10;
usp0=sp20;
usp1=sp20;
while (1)
{
    d='2';
    putchar(d);
    if ((d=getchar())=='8')
    {
        d=getchar();
        initstru();
        compute();
    }
}

```



```

senddata();
eot();
sleep(120);
}
}
}

initstru()
{
/* this a procedure to initialize structure to test module stand-alone. */
int fix0,fiy0,fix1,fiy1,fix2,fiy2;
int hix0,hiy0,hix1,hiy1,hix2,hiy2,hix3,hiy3;
int uix0,uiy0,uix1,uiy1;
shippot[0].speed[0]=fsp0;
shippot[0].speed[1]=fsp1;
shippot[0].speed[2]=fsp2;
shippot[1].speed[0]=hsp0;
shippot[1].speed[1]=hsp1;
shippot[1].speed[2]=hsp2;
shippot[1].speed[3]=hsp3;
shippot[2].speed[0]=usp0;
shippot[2].speed[1]=usp1;
shippot[0].count=3;
shippot[1].count=4;
shippot[2].count=2;
shippot[0].xnow[0]=fx0;
shippot[0].xbow[1]=fx1;

```



```

shipp[0].xbow[2]=fx2;
shipp[1].xbow[0]=hx0;
shipp[1].xbow[1]=hx1;
shipp[1].xbow[2]=hx2;
shipp[1].xbow[3]=hx3;
shipp[2].xbow[0]=ux0;
shipp[2].xbow[1]=ux1;
shipp[0].ybow[0]=fy0;
shipp[0].ybow[1]=fy1;
shipp[0].ybow[2]=fy2;
shipp[1].ybow[0]=hy0;
shipp[1].ybow[1]=hy1;
shipp[1].ybow[2]=hy2;
shipp[1].ybow[3]=hy3;
shipp[2].ybow[0]=uy0;
shipp[2].ybow[1]=uy1;
shipp[0].quadrant[0]=quadnum(shipp[0].xbow[0],shipp[0].ybow[0]);
shipp[0].quadrant[1]=quadnum(shipp[0].xbow[1],shipp[0].ybow[1]);
shipp[0].quadrant[2]=quadnum(shipp[0].xbow[2],shipp[0].ybow[2]);
shipp[1].quadrant[0]=quadnum(shipp[1].xbow[0],shipp[1].ybow[0]);
shipp[1].quadrant[1]=quadnum(shipp[1].xbow[1],shipp[1].ybow[1]);
shipp[1].quadrant[2]=quadnum(shipp[1].xbow[2],shipp[1].ybow[2]);
shipp[1].quadrant[3]=quadnum(shipp[1].xbow[3],shipp[1].ybow[3]);
shipp[2].quadrant[0]=quadnum(shipp[2].xbow[0],shipp[2].ybow[0]);
shipp[2].quadrant[1]=quadnum(shipp[2].xbow[1],shipp[2].ybow[1]);
shipp[2].quadrant[2]=quadnum(shipp[2].xbow[2],shipp[2].ybow[2]);
shipp[2].quadrant[3]=quadnum(shipp[2].xbow[3],shipp[2].ybow[3]);

/* increment data points for all contacts */
if (intct < 8)
{
    fix0=fix1=fix2=2;

```



```

    fix0=fix1=fix2= -1;
    hix0=hix1=hix2=hix3=1;
    hiy0=hiy1=hiy2=hiy3= -1;
    uix0=0;
    uiy0=2;
    uix1= -1;
    uiy1= -2;
}
else
if (inter < 22)
{
    fix0=fix1=fix2=0;
    fix0=fix1=fix2= -3;
    hix0=hix1=hix2=hix3=0;
    hiy0=hiy1=hiy2=hiy3= -2;
    uix0=0;
    uiy0=2;
    uix1= -1;
    uiy1= -2;
    hsp0=hsp1=hsp2=hsp3=sp20;
    fsp0=fsp1=fsp2=fsp30;
}
else
{
    fix0=fix1=fix2= -3;
    fix0=fix1=fix2= -3;
    hix0=hix1=hix2=hix3= -5;
    hiy0=hiy1=hiy2=hiy3= -3;
    uix0=0;
    uiy0=2;

```



```

uix1= -1;
uiy1= -2;
fsp0=fsp1=fsp2=40;
hsp0=hsp1=hsp2=hsp3=30;
}

```

```

fx0=fix0 + fx0;
fy0=fiy0 + fy0;
fx1=fix1 + fx1;
fy1=fiy1 + fy1;
fx2=fix2 + fx2;
fy2=fiy2 + fy2;
hx0=hix0 + hx0;
hy0=hiy0 + hy0;
hx1=hix1 + hx1;
hy1=hiy1 + hy1;
hx2=hix2 + hx2;
hy2=hiy2 + hy2;
hx3=hix3 + hx3;
hy3=hiy3 + hy3;
ux0=uix0 + ux0;
uy0=uiy0 + uy0;
ux1=uix1 + ux1;
uy1=uiy1 + uy1;

intct = intct + (times * 1);
} /* initstru */

```



```

init0()
{
    char d,e;
    for (e=0;e<3;e++)
    {
        shipplot[el].count=0;
        for(d=0;d<10;d++)
        {
            shipplot[el].lat[d]=0;
            shipplot[el].longtd[d]=0;
            shipplot[el].course[d]=0;
            shipplot[el].speed[d]=0;
            shipplot[el].xbow[d]=0;
            shipplot[el].ybow[d]=0;
            shipplot[el].quadrant[d]=0;
            shipplot[el].range[d]=0;
            shipplot[el].bearing[d]=0;
            shipplot[el].colflag[d]=0;
            shipplot[el].cpatime[d]=0;
            shipplot[el].cpadist[d]=0;
        }
    }
}

senddata()
{
    char d,e;

```



```

d = '0';
putchar(d);
for (e=0;e<3;e++)
{
    for(d=0;d<10;d++)
    {
        if ((shipplot[e].lat[d] & bitx7 ) > 0)
            shipplot[e].lat[d] = (shipplot[e].lat[d] ; bitx14);
        z=(shipplot[e].lat[d]);
        write(1,8z,1);
        z=(shipplot[e].lat[d]>>8);
        write(1,8z,1);
    }
    for(d=0;d<10;d++)
    {
        if ((shipplot[e].longtd[d] & bitx7) > 0)
            shipplot[e].longtd[d] = (shipplot[e].longtd[d] ; bitx14);
        z=(shipplot[e].longtd[d]);
        write(1,8z,1);
        z=(shipplot[e].longtd[d]>>8);
        write(1,8z,1);
    }
    for(d=0;d<10;d++)
    {
        if ((shipplot[e].course[d] & bitx7) > 0)
            shipplot[e].course[d] = (shipplot[e].course[d] ; bitx14);
        z=(shipplot[e].course[d]);
        write(1,8z,1);
        z=(shipplot[e].course[d]>>8);
        write(1,8z,1);
    }
}

```



```

    }
    for(d=0;d<10;d++)
    {
        z=(shipplot[el].speed[d]);
        write(1,&z,1);
    }
    for(d=0;d<10;d++)
    {
        if ((shipplot[el].xbow[d] & bitx7) > 0)
            shipplot[el].xbow[d] = (shipplot[el].xbow[d] ; bitx14);
        z=(shipplot[el].xbow[d]);
        write(1,&z,1);
        z = (shipplot[el].xbow[d]>>8);
        write(1,&z,1);
    }
    for(d=0;d<10;d++)
    {
        if ((shipplot[el].ybow[d] & bitx7) > 0)
            shipplot[el].ybow[d] = (shipplot[el].ybow[d] ; bitx14);
        z=(shipplot[el].ybow[d]);
        write(1,&z,1);
        z = (shipplot[el].ybow[d]>>8);
        write(1,&z,1);
    }
    for(d=0;d<10;d++)
    {
        z=(shipplot[el].quadrant[d]);
        write(1,&z,1);
    }
    for(d=0;d<10;d++)

```



```

{
    if ((shipplot[el].range[d] & bitx7) > 0)
        shipplot[el].range[d] = (shipplot[el].range[d] ; bitx14);
    z=(shipplot[el].range[d]);
    write(1,8z,1);
    z = (shipplot[el].range[d]>>8);
    write(1,8z,1);
}
for(d=0;d<10;d++)
{
    if ((shipplot[el].bearing[d] & bitx7) > 0)
        shipplot[el].bearing[d] = (shipplot[el].bearing[d] ; bitx14);
    z=(shipplot[el].bearing[d]);
    write(1,8z,1);
    z = (shipplot[el].bearing[d]>>8);
    write(1,8z,1);
}
for(d=0;d<10;d++)
{
    z=(shipplot[el].collisionflag[d]);
    write(1,8z,1);
}
for(d=0;d<10;d++)
{
    if ((shipplot[el].cpatime[d] & bitx7) > 0)
        shipplot[el].cpatime[d] = (shipplot[el].cpatime[d] ; bitx14);
    z=(shipplot[el].cpatime[d]);
    write(1,8z,1);
    z = (shipplot[el].cpatime[d]>>8);
    write(1,8z,1);
}

```



```

    }
    for(d=0;d<10;d++)
    {
        if ((shipplot[el].cpadist[d] & bitx7) > 0)
            shipplot[el].cpadist[d] = (shipplot[el].cpadist[d] ! bitx14);
        z=(shipplot[el].cpadist[d]);
        write(1,&z,1);
        z = (shipplot[el].cpadist[d]>>8);
        write(1,&z,1);
    }
    z=(shipplot[el].count);
    write(1,&z,1);
}

init30()
{
    char d,e;
    for (e=0;e<3;e++)
    {
        shipplot[el].count=0;
        for(d=0;d<10;d++)
        {
            shipplot[el].lat[d]=030062;
            shipplot[el].longtd[d]=030062;
            shipplot[el].course[d]=030062;
            shipplot[el].speed[d]=061;
            shipplot[el].xbow[d]=030062;
            shipplot[el].yhow[d]=030062;
        }
    }
}

```



```

    shipplot[el].quadrant[el]=061;
    shipplot[el].range[el]=061;
    shipplot[el].bearing[el]=030062;
    shipplot[el].collflag[el]=061;
    shipplot[el].cpatime[el]=030062;
    shipplot[el].cpadist[el]=030062;
}
}

eot()
{
    putchar(cr);
    putchar(lf);
    putchar('%');
}

quadnum(x,y)
int x,y;
{
    char a;
    if ((x>=0) && (x<=127))
    {
        if ((y>=0) && (y<=127))
            a=1;
        else

```



```
if ((y>=128) && (y<=255))
    a=5;
else
    if ((y>=256) && (y<=383))
        a=9;
    else
        if ((y>=384) && (y<=511))
            a=13;
        }
    else
        if ((x>=128) && (x<=255))
        {
            if ((y>=0) && (y<=127))
                a=2;
            else
                if ((y>=128) && (y<=255))
                    a=6;
                else
                    if ((y>=256) && (y<=383))
                        a=10;
                    else
                        if ((y>=384) && (y<=511))
                            a=14;
                }
            else
                if ((x>=256) && (x<=383))
                {
                    if ((y>=0) && (y<=127))
                        a=3;
```



```

else
if ((y>=128) && (y<=255))
    a=7;
else
if ((y>=256) && (y<=383))
    a=11;
else
if ((y>=384) && (y<=511))
    a=15;
}
else
if ((x>=384) && (x<=511))
{
    if ((y>=0) && (y<=127))
        a=4;
    else
if ((y>=128) && (y<=255))
        a=8;
    else
if ((y>=256) && (y<=383))
        a=12;
    else
if ((y>=384) && (y<=511))
        a=16;
}

return a;
} /* quadnum */

```



```

initck()
{
    char d,e;
    for (e=0;e<3;e++)
    {
        shippilot[el].count='c';
        for(d=0;d<10;d++)
        {
            shippilot[el].lat[d]='00';
            shippilot[el].lonat[d]='11';
            shippilot[el].course[d]='22';
            shippilot[el].speed[d]='3';
            shippilot[el].xhow[d]='44';
            shippilot[el].yhow[d]='55';
            shippilot[el].quadrant[d]='6';
            shippilot[el].range[d]='77';
            shippilot[el].bearing[d]='88';
            shippilot[el].collflag[d]='9';
            shippilot[el].cpatime[d]='aa';
            shippilot[el].cpadist[d]='bb';
        }
    }
}

```



```

compute()
/* calls subroutines to load contact statistical data */
{
    int i,j ;
    i=0 ;
    for (i=0;i<3;i++)
    {
        if (shipplotfil.count > 0)
        {
            while(j<shipplotfil.count)
            {
                ldlat(i,j) ;
                ldlong(i,j) ;
                ldcour(i,j) ;
                if ((i==0) && (j==1)) j=j ;
                else
                {
                    ldrang(i,j) ;
                    ldbear(i,j) ;
                }
                j=j+1 ;
            }
            j=0 ;
        }
    }
}

```



```

ldlat(i,j)
{
    /* load latitude of contact */
    float fy,flat ;
    int ilat ;
    fy=shipplot(il.xbow[j] * 1.0 ;
    flat=baselat + (fy * latoft) ;
    flat=flat * 10.0;
    ilat=flat ;
    /* set flag to north */
    shipplot(il.lat[j]=ilat ; 020000 ;
}

```

```

ldlong(i,j)
{
    /* load longitude of contact */
    float fx,flong ;
    int ilong ;
    fx=shipplot(il.xbow[j] * 1.0 ;
    flong=baselong + (fx * longoff) ;
    flong =flong * 10.0;
}

```



```

    ilong=flong;
    /* set flag to west */
    shipplotfil.longtd[j] = ilong ; 020000 ;
}

longang(i,j)

int i,j ;
{
    /* load contact range */
    float r1,t1,r2,t2,frange ;
    int irange ;
    r1=shipplot[0].xhow[1] * 1.0 ;
    t1=shipplot[0].yhow[1] * 1.0 ;
    r2=shipplot[i].xhow[j] * 1.0 ;
    t2=shipplot[i].yhow[j] * 1.0 ;
    frange= sqrt(pow((r2-r1),2.0) + pow((t2-t1),2.0));
    frange=frange * 312.5 ;
    frange=frange / 10.0 ;
    irange=frange ;
    shipplotfil.range[j]=irange ;
}

canqle(xh,yh,xt,yt)

```



```

int xh,yh,xt,yt;
{
/* calculate angle given head and tail points */
int iangle;
float fxh,fyh,fxh,fxh,fyt,fangle,arb,ftan;
fxh=xh;
fyh=yh;
fyt=yt;
fxh=xt;
a= fabs(fyt-fyh);
h= fabs(fxt-fxh);
ftan = b / a;
fangle = atan(ftan);
fangle=fangle * 57.3;
iangle = fangle;
return iangle;
}

lucour(i,j)

int i,j;
{
/* loads contact course */
int xh,yh,xt,yt,alpha;
xh = shipplot[i].xhowl[j];

```



```

yh = shipplot[i].yhowl[j];
xt = back[i].xtail[j];
yt = back[i].ytail[j];
if ((xt!= 1000) && (!((xh == xt) && (yh==yt))))
{
    if (xh == xt)
    {
        if (yh<yt) alpha = 0;
        else alpha = 180;
    }
    else if (yh==yt)
    {
        if (xh>xt) alpha = 90;
        else alpha = 270;
    }
    else if (xh>xt)
    {
        if (yh<yt) alpha = cangle(xh,yh,xt,yt);
        else alpha = 180 - cangle(xh,yh,xt,yt);
    }
    else if (xh<xt)
    {
        if (yh<yt) alpha = 360 - cangle(xh,yh,xt,yt);
        else alpha = 180 + cangle(xh,yh,xt,yt);
    }
    if (alpha==360) alpha=0;
    shipplot[i].course[j] = alpha;
} /* if (xt != 1000) */
else shipplot[i].course[j] = 0;
back[i].xtail[j] = xh;

```



```

back[i].vtail[j] = yh;
} /* ld cour (i,j) */

ldhear(i,j)

int i,j ;
{
    /* load contact bearing */
    int xs,ys,xc,yc,alpha,ibear,tbear,owncou,tempb;
    xc = shipplot[i].xhow[j];
    yc = shipplot[i].yhow[j];
    xs = shipplot[0].xhow[j];
    ys = shipplot[0].yhow[j];
    if (xc==xs)
    {
        if (yc<ys) alpha = 0;
        else alpha = 180;
    }
    else if (yc==ys)
    {
        if (xc>xs) alpha = 90;
        else alpha = 270;
    }
    else if (xc>xs)
    {
        if (yc<ys) alpha = cangle(xc,yc,xs,ys);
        else alpha = 180 - cangle(xc,yc,xs,ys);
    }
}

```



```

    }
    else if (xc<xs)
    {
        if (yc<ys) alpha = 360 - cangle(xc,yc,xs,ys);
        else alpha = 180 + cangle(xc,yc,xs,ys);
    }

    owncon = shipplot[0].course[1];
    tbear = alpha - owncon;
    if (tbear < (-180)) tbear = 360 + tbear;
    else if (tbear > 180) tbear = -360 + tbear;
    temph = tbear;
    ibear = abs(tbear);
    if (temph>=0)
        ibear = (ibear ! 020000); /* set s */
    shipplot[i].bearing[ij] = ibear;
}

ldcpatd(i,j)

int i,j;
{
    /* this procedure loads cpa time and cpa distance to structure */
    int icpat,icpad,imin;
    float x0,y0,x1,y1,x2,y2,m,lbear,nbear,drm,xcpa,ycpa;
    float cpat,cpat,rang1,rang2 ,ospeed,x,y,thour,fmin;
    if ((backli.bear[ij] == 1000) || (cpact == 4))
    {

```



```

rang1=(backfil.ranqtailjl * 10.0) ;
rang2=(shipplotfil.rangefjl * 10.0) ;
if ((backfil.beartailjl & 020000) > 0)
    lbear= ((backfil.beartailjl & 057777) * 1.0) / 57.3 ;
else
    lbear=(360.0 - (backfil.beartailjl * 1.0)) / 57.3 ;
if ((shipplotfil.bearingljl & 020000) > 0)
    nbear=((shipplotfil.bearingljl & 057777) * 1.0) / 57.3 ;
else
    nbear=(360.0 - (shipplotfil.bearingljl * 1.0)) / 57.3 ;
ospeed=shipplotfil.speedfjl * 1.0 * 2000.0;
if (backfil.beartailjl != 1000)
{
    x1=rang1 * (cos(lbear));
    y1=rang1 * (sin(lbear));
    x2=rang2 * (cos(nbear));
    y2=rang2 * (sin(nbear));
    x0=x2-x1;
    y0=y2-y1;
    if ((fabs(y0) - fabs(x0)) >= 0)
    {
        v=x0;
        x = -y0;
        x1 = -y1;
        x2 = -y2;
        y1=x1;
    }
    else
    {
        x=x0;

```



```

y=y0;
}
m=y/x;
drr=atan(m);
xcpa=((x1*pow(m,2)) - (y1*m)) / (pow(m,2) + 1.0);
ycpa=(y1 - (x1 * m)) / (pow(m,2) + 1.0);
cpad=sqrt(pow(xcpa,2) + pow(ycpa,2));
cpat=sqrt(pow((xcpa-x1),2) + pow((ycpa-y1),2)) / ospeed ;
cpad=cpad / 10.0;
icpad=fabs(cpad);
if (icpad > 16000) icpad=0;
if (cpat < 0 ) cpat=0.0;
if (cpat > 24) cpat=0.0;
icpat=fabs(cpat);
thour=icpat * 1.0;
icpat=icpat * 100;
fmin=fabs(cpat) - thour;
fmin=((fmin * 10.0) * 6.0 );
imin=fmin;
icpat=icpat + imin;
shipplot[i].cpatime[j]=icpat;
shipplot[i].cpadist[j]=icpad;
}
else
{
shipplot[i].cpatime[j]=0;
shipplot[i].cpadist[j]=0;
}
backfil.rangtail[j]=shipplot[i].range[j];
backfil.heartail[j]=shipplot[i].bearing[j];

```



```
cpact = 0;  
}  
else  
{  
    cpact = cpact + 1;  
}  
} /* ldenatd */
```



```

2 1          /* DECLARATIONS: **** */
3 1  DECLARE LIT LITERALLY 'LITERALLY';
4 1  DECLARE DCL LIT 'DECLARE';
5 1          /* ISIS-II SYSTEM CONSTANTS **/
6 1  DCL TOP LIT '0F800H'; /* TOP OF FREE SPACE */
7 1  DCL BASE LIT '0D400H'; /* BASE OF PDP BUFFER */
8 1          /* PROGRAM CONSTANTS: **** */
9 1  DCL TRUE LIT '0FFH';

```



```

7      1      DCL      FALSE      LIT      '000H';
8      1      DCL      FOREVER     LIT      'WHILE 1';
9      1      DCL      RCVD        LIT      '2EH';
10     1      DCL      NEUTRAL      LIT      '0';
11     1      DCL      RECEIVE     LIT      '1';
12     1      DCL      TRANSMIT     LIT      '2';
13     1      DCL      ACK          LIT      '06H';
14     1      DCL      READ$ACCESS  LIT      '1';
15     1      DCL      END$OF$BLOCK LIT      '01H';

```

/* GLOBAL VARIABLES: *****/

```

16     1      DCL      CRT$BUFFER (200) BYTE;
17     1      DCL      FILE$NAME (128) BYTE;
18     1      DCL      PDP$BUF$ADDRESS ADDRESS INITIAL (BASE);
19     1      DCL      PDP$BUFFER BASED PDP$BUF$ADDRESS (9000)
BYTE;

```



```

20 1      DCL ( PDP$BUF$FIRST,
              PDP$BUF$LAST,
              PDP$BUF$NUMBER,
              CRT$BUF$FIRST,
              CRT$BUF$LAST,
              CRT$BUF$NUMBER,
              LENGTH$PDP$BUF,
              LAST$LOC$PDP$BUF,
              WRITE$STATUS,
              DELETE$STATUS,
              READ$STATUS,
              OPEN$STATUS,
              PDP$BUF$PTR,
              CLOSE$STATUS,
              CONSOL$STATUS,
              BYTE$COUNT,
              COUNT$TEMP,
              AFT$IN ) ADDRESS ;

21 1      DCL ( MISSINGDATA,
              STOP$FLAG,
              SENT$ACK,

```



```
ALARM,  
D1,D2,D3,  
CHAR$TWO,  
CHAR,  
STATE,  
PREVCHAR,  
EOFILF,  
SKIP$FIVE,  
ECHOCHAR,  
SAVECHAR ) BYTE ;  
22 DCL CRTECHO BYTE INITIAL (0);  
23 DCL PDPECHO BYTE INITIAL (0);
```

/* SPECIAL CHARACTERS: *****/

```
24 1 DCL RUBOUT LIT '7FH',  
PROMPT LIT '25H', /* PERCENT */
```



```

CONTROL$W LIT '17H',
CONTROL$N LIT '4EH',
CONTROL$R LIT '12H',
CONTROL$T LIT '14H',
CONTROL$Z LIT '1AH',
CONTROL$C LIT '03H',
CR        LIT '0DH',
LF        LIT '0AH',
BELL      LIT '07H';

```

/*ISIS-II SYSTEM CALLS: *****/

```

25 1  OPEN:
      PROCEDURE (AFT,FILE,ACCESS,MODE,STATUS ) EXTER
      DECLARE (AFT,FILE,ACCESS,MODE,STATUS )AD
26 2  NAL;
27 2  DRESS;
      END OPEN;

```



```
28 1      CLOSE:
29 2      PROCEDURE(AFT,STATUS) EXTERNAL;
30 2      DCL (AFT,STATUS) ADDRESS;
      END CLOSE;

31 1      READ:
      PROCEDURE(AFT,BUFFER,COUNT,ACTUAL,STATUS) EXTE
RNAL;
32 2      DCL(AFT,BUFFER,COUNT,ACTUAL,STATUS) ADDRESS;
33 2      END READ;

34 1      WRITE:
      PROCEDURE(AFT,BUFFER,COUNT,STATUS) EXTERNAL;
35 2      DCL(AFT,BUFFER,COUNT,STATUS) ADDRESS;
36 2      END WRITE;
```



```

37 1      EXIT:
38 2          PROCEDURE EXTERNAL;
39 2          DCL STATUS ADDRESS;
          END EXIT;

40 1      CONSOL:
41 2          PROCEDURE(INFILE,OUTFILE,STATUS) EXTERNAL;
42 2          DCL (INFILE,OUTFILE,STATUS) ADDRESS;
          END CONSOL;

43 1      DELETE:
44 2          PROCEDURE(FILE,STATUS) EXTERNAL;
45 2          DCL(FILE,STATUS) ADDRESS;
          END DELETE;

```



```
46 1      ERROR:
47 2          PROCEDURE (ERRNUM) EXTERNAL;
48 2          DCL (ERRNUM) ADDRESS;
      END ERROR;

49 1      RENAME:
50 2          PROCEDURE(OLDFILE,NEWFILE,STATUS) EXTERNAL;
51 2          DCL(OLDFILE,NEWFILE,STATUS) ADDRESS;
      END RENAME;
```

```
/* PROCEDURES:*****//
```



```

52 1      SET$TTY$2400: PROCEDURE;

53 2      /* SET TTY BAUD RATE TO 2400 */
54 2      OUTPUT(245)=40H;
55 2      OUTPUT(245)=4FH;
56 2      OUTPUT(245)=37H;
56 2      END;

57 1      OUTPUT$STATUS$CRT: PROCEDURE BYTE;

58 2      /* TRUE IF DATA OUTPUT LINE TO CRT READY */
59 2      RETURN FOR (INPUT(247),2);
59 2      END;

```



```
60 1      OUTPUT$STATUS$PDP: PROCEDURE BYTE;
61 2      /* TRUE IF DATA OUTPUT LINE TO PDP READY */
62 2      RETURN ROR(INPUT(245),2);
        END;

63 1      INPUT$STATUS$CRT: PROCEDURE BYTE;
64 2      /* TRUE IF DATA INPUT LINE FROM CRT READY */
65 2      RETURN ROR(INPUT(247),1);
        END;

66 1      INPUT$STATUS$PDP: PROCEDURE BYTE ;
67 2      /* TRUE IF DATA INPUT LINE FROM PDP READY */
68 2      RETURN ROR(INPUT(245),1) ;
        END;
```



```
69 1      SEND$CHAR$CRT: PROCEDURE (CHAR) ;
    2
    3      /* PRINT A CHARACTER TO THE CRT */
    4      DCL CHAR BYTE;
    5      DO WHILE NOT OUTPUT$STATUS$CRT;
    6          END;
    7      OUTPUT(246)= CHAR;
    8      END;
    9
    10
    11
    12
    13
    14
    15
    16
    17
    18
    19
    20
    21
    22
    23
    24
    25
    26
    27
    28
    29
    30
    31
    32
    33
    34
    35
    36
    37
    38
    39
    40
    41
    42
    43
    44
    45
    46
    47
    48
    49
    50
    51
    52
    53
    54
    55
    56
    57
    58
    59
    60
    61
    62
    63
    64
    65
    66
    67
    68
    69
    70
    71
    72
    73
    74
    75
    76
    77
    78
    79
    80
    81
    82
    83
    84
    85
    86
    87
    88
    89
    90
    91
    92
    93
    94
    95
    96
    97
    98
    99
    100
    101
    102
    103
    104
    105
    106
    107
    108
    109
    110
    111
    112
    113
    114
    115
    116
    117
    118
    119
    120
    121
    122
    123
    124
    125
    126
    127
    128
    129
    130
    131
    132
    133
    134
    135
    136
    137
    138
    139
    140
    141
    142
    143
    144
    145
    146
    147
    148
    149
    150
    151
    152
    153
    154
    155
    156
    157
    158
    159
    160
    161
    162
    163
    164
    165
    166
    167
    168
    169
    170
    171
    172
    173
    174
    175
    176
    177
    178
    179
    180
    181
    182
    183
    184
    185
    186
    187
    188
    189
    190
    191
    192
    193
    194
    195
    196
    197
    198
    199
    200
    201
    202
    203
    204
    205
    206
    207
    208
    209
    210
    211
    212
    213
    214
    215
    216
    217
    218
    219
    220
    221
    222
    223
    224
    225
    226
    227
    228
    229
    230
    231
    232
    233
    234
    235
    236
    237
    238
    239
    240
    241
    242
    243
    244
    245
    246
    247
    248
    249
    250
    251
    252
    253
    254
    255
    256
    257
    258
    259
    260
    261
    262
    263
    264
    265
    266
    267
    268
    269
    270
    271
    272
    273
    274
    275
    276
    277
    278
    279
    280
    281
    282
    283
    284
    285
    286
    287
    288
    289
    290
    291
    292
    293
    294
    295
    296
    297
    298
    299
    300
    301
    302
    303
    304
    305
    306
    307
    308
    309
    310
    311
    312
    313
    314
    315
    316
    317
    318
    319
    320
    321
    322
    323
    324
    325
    326
    327
    328
    329
    330
    331
    332
    333
    334
    335
    336
    337
    338
    339
    340
    341
    342
    343
    344
    345
    346
    347
    348
    349
    350
    351
    352
    353
    354
    355
    356
    357
    358
    359
    360
    361
    362
    363
    364
    365
    366
    367
    368
    369
    370
    371
    372
    373
    374
    375
    376
    377
    378
    379
    380
    381
    382
    383
    384
    385
    386
    387
    388
    389
    390
    391
    392
    393
    394
    395
    396
    397
    398
    399
    400
    401
    402
    403
    404
    405
    406
    407
    408
    409
    410
    411
    412
    413
    414
    415
    416
    417
    418
    419
    420
    421
    422
    423
    424
    425
    426
    427
    428
    429
    430
    431
    432
    433
    434
    435
    436
    437
    438
    439
    440
    441
    442
    443
    444
    445
    446
    447
    448
    449
    450
    451
    452
    453
    454
    455
    456
    457
    458
    459
    460
    461
    462
    463
    464
    465
    466
    467
    468
    469
    470
    471
    472
    473
    474
    475
    476
    477
    478
    479
    480
    481
    482
    483
    484
    485
    486
    487
    488
    489
    490
    491
    492
    493
    494
    495
    496
    497
    498
    499
    500
    501
    502
    503
    504
    505
    506
    507
    508
    509
    510
    511
    512
    513
    514
    515
    516
    517
    518
    519
    520
    521
    522
    523
    524
    525
    526
    527
    528
    529
    530
    531
    532
    533
    534
    535
    536
    537
    538
    539
    540
    541
    542
    543
    544
    545
    546
    547
    548
    549
    550
    551
    552
    553
    554
    555
    556
    557
    558
    559
    560
    561
    562
    563
    564
    565
    566
    567
    568
    569
    570
    571
    572
    573
    574
    575
    576
    577
    578
    579
    580
    581
    582
    583
    584
    585
    586
    587
    588
    589
    590
    591
    592
    593
    594
    595
    596
    597
    598
    599
    600
    601
    602
    603
    604
    605
    606
    607
    608
    609
    610
    611
    612
    613
    614
    615
    616
    617
    618
    619
    620
    621
    622
    623
    624
    625
    626
    627
    628
    629
    630
    631
    632
    633
    634
    635
    636
    637
    638
    639
    640
    641
    642
    643
    644
    645
    646
    647
    648
    649
    650
    651
    652
    653
    654
    655
    656
    657
    658
    659
    660
    661
    662
    663
    664
    665
    666
    667
    668
    669
    670
    671
    672
    673
    674
    675
    676
    677
    678
    679
    680
    681
    682
    683
    684
    685
    686
    687
    688
    689
    690
    691
    692
    693
    694
    695
    696
    697
    698
    699
    700
    701
    702
    703
    704
    705
    706
    707
    708
    709
    710
    711
    712
    713
    714
    715
    716
    717
    718
    719
    720
    721
    722
    723
    724
    725
    726
    727
    728
    729
    730
    731
    732
    733
    734
    735
    736
    737
    738
    739
    740
    741
    742
    743
    744
    745
    746
    747
    748
    749
    750
    751
    752
    753
    754
    755
    756
    757
    758
    759
    760
    761
    762
    763
    764
    765
    766
    767
    768
    769
    770
    771
    772
    773
    774
    775
    776
    777
    778
    779
    780
    781
    782
    783
    784
    785
    786
    787
    788
    789
    790
    791
    792
    793
    794
    795
    796
    797
    798
    799
    800
    801
    802
    803
    804
    805
    806
    807
    808
    809
    810
    811
    812
    813
    814
    815
    816
    817
    818
    819
    820
    821
    822
    823
    824
    825
    826
    827
    828
    829
    830
    831
    832
    833
    834
    835
    836
    837
    838
    839
    840
    841
    842
    843
    844
    845
    846
    847
    848
    849
    850
    851
    852
    853
    854
    855
    856
    857
    858
    859
    860
    861
    862
    863
    864
    865
    866
    867
    868
    869
    870
    871
    872
    873
    874
    875
    876
    877
    878
    879
    880
    881
    882
    883
    884
    885
    886
    887
    888
    889
    890
    891
    892
    893
    894
    895
    896
    897
    898
    899
    900
    901
    902
    903
    904
    905
    906
    907
    908
    909
    910
    911
    912
    913
    914
    915
    916
    917
    918
    919
    920
    921
    922
    923
    924
    925
    926
    927
    928
    929
    930
    931
    932
    933
    934
    935
    936
    937
    938
    939
    940
    941
    942
    943
    944
    945
    946
    947
    948
    949
    950
    951
    952
    953
    954
    955
    956
    957
    958
    959
    960
    961
    962
    963
    964
    965
    966
    967
    968
    969
    970
    971
    972
    973
    974
    975
    976
    977
    978
    979
    980
    981
    982
    983
    984
    985
    986
    987
    988
    989
    990
    991
    992
    993
    994
    995
    996
    997
    998
    999
    1000
    1001
    1002
    1003
    1004
    1005
    1006
    1007
    1008
    1009
    1010
    1011
    1012
    1013
    1014
    1015
    1016
    1017
    1018
    1019
    1020
    1021
    1022
    1023
    1024
    1025
    1026
    1027
    1028
    1029
    1030
    1031
    1032
    1033
    1034
    1035
    1036
    1037
    1038
    1039
    1040
    1041
    1042
    1043
    1044
    1045
    1046
    1047
    1048
    1049
    1050
    1051
    1052
    1053
    1054
    1055
    1056
    1057
    1058
    1059
    1060
    1061
    1062
    1063
    1064
    1065
    1066
    1067
    1068
    1069
    1070
    1071
    1072
    1073
    1074
    1075
    1076
    1077
    1078
    1079
    1080
    1081
    1082
    1083
    1084
    1085
    1086
    1087
    1088
    1089
    1090
    1091
    1092
    1093
    1094
    1095
    1096
    1097
    1098
    1099
    1100
    1101
    1102
    1103
    1104
    1105
    1106
    1107
    1108
    1109
    1110
    1111
    1112
    1113
    1114
    1115
    1116
    1117
    1118
    1119
    1120
    1121
    1122
    1123
    1124
    1125
    1126
    1127
    1128
    1129
    1130
    1131
    1132
    1133
    1134
    1135
    1136
    1137
    1138
    1139
    1140
    1141
    1142
    1143
    1144
    1145
    1146
    1147
    1148
    1149
    1150
    1151
    1152
    1153
    1154
    1155
    1156
    1157
    1158
    1159
    1160
    1161
    1162
    1163
    1164
    1165
    1166
    1167
    1168
    1169
    1170
    1171
    1172
    1173
    1174
    1175
    1176
    1177
    1178
    1179
    1180
    1181
    1182
    1183
    1184
    1185
    1186
    1187
    1188
    1189
    1190
    1191
    1192
    1193
    1194
    1195
    1196
    1197
    1198
    1199
    1200
    1201
    1202
    1203
    1204
    1205
    1206
    1207
    1208
    1209
    1210
    1211
    1212
    1213
    1214
    1215
    1216
    1217
    1218
    1219
    1220
    1221
    1222
    1223
    1224
    1225
    1226
    1227
    1228
    1229
    1230
    1231
    1232
    1233
    1234
    1235
    1236
    1237
    1238
    1239
    1240
    1241
    1242
    1243
    1244
    1245
    1246
    1247
    1248
    1249
    1250
    1251
    1252
    1253
    1254
    1255
    1256
    1257
    1258
    1259
    1260
    1261
    1262
    1263
    1264
    1265
    1266
    1267
    1268
    1269
    1270
    1271
    1272
    1273
    1274
    1275
    1276
    1277
    1278
    1279
    1280
    1281
    1282
    1283
    1284
    1285
    1286
    1287
    1288
    1289
    1290
    1291
    1292
    1293
    1294
    1295
    1296
    1297
    1298
    1299
    1300
    1301
    1302
    1303
    1304
    1305
    1306
    1307
    1308
    1309
    1310
    1311
    1312
    1313
    1314
    1315
    1316
    1317
    1318
    1319
    1320
    1321
    1322
    1323
    1324
    1325
    1326
    1327
    1328
    1329
    1330
    1331
    1332
    1333
    1334
    1335
    1336
    1337
    1338
    1339
    1340
    1341
    1342
    1343
    1344
    1345
    1346
    1347
    1348
    1349
    1350
    1351
    1352
    1353
    1354
    1355
    1356
    1357
    1358
    1359
    1360
    1361
    1362
    1363
    1364
    1365
    1366
    1367
    1368
    1369
    1370
    1371
    1372
    1373
    1374
    1375
    1376
    1377
    1378
    1379
    1380
    1381
    1382
    1383
    1384
    1385
    1386
    1387
    1388
    1389
    1390
    1391
    1392
    1393
    1394
    1395
    1396
    1397
    1398
    1399
    1400
    1401
    1402
    1403
    1404
    1405
    1406
    1407
    1408
    1409
    1410
    1411
    1412
    1413
    1414
    1415
    1416
    1417
    1418
    1419
    1420
    1421
    1422
    1423
    1424
    1425
    1426
    1427
    1428
    1429
    1430
    1431
    1432
    1433
    1434
    1435
    1436
    1437
    1438
    1439
    1440
    1441
    1442
    1443
    1444
    1445
    1446
    1447
    1448
    1449
    1450
    1451
    1452
    1453
    1454
    1455
    1456
    1457
    1458
    1459
    1460
    1461
    1462
    1463
    1464
    1465
    1466
    1467
    1468
    1469
    1470
    1471
    1472
    1473
    1474
    1475
    1476
    1477
    1478
    1479
    1480
    1481
    1482
    1483
    1484
    1485
    1486
    1487
    1488
    1489
    1490
    1491
    1492
    1493
    1494
    1495
    1496
    1497
    1498
    1499
    1500
    1501
    1502
    1503
    1504
    1505
    1506
    1507
    1508
    1509
    1510
    1511
    1512
    1513
    1514
    1515
    1516
    1517
    1518
    1519
    1520
    1521
    1522
    1523
    1524
    1525
    1526
    1527
    1528
    1529
    1530
    1531
    1532
    1533
    1534
    1535
    1536
    1537
    1538
    1539
    1540
    1541
    1542
    1543
    1544
    1545
    1546
    1547
    1548
    1549
    1550
    1551
    1552
    1553
    1554
    1555
    1556
    1557
    1558
    1559
    1560
    1561
    1562
    1563
    1564
    1565
    1566
    1567
    1568
    1569
    1570
    1571
    1572
    1573
    1574
    1575
    1576
    1577
    1578
    1579
    1580
    1581
    1582
    1583
    1584
    1585
    1586
    1587
    1588
    1589
    1590
    1591
    1592
    1593
    1594
    1595
    1596
    1597
    1598
    1599
    1600
    1601
    1602
    1603
    1604
    1605
    1606
    1607
    1608
    1609
    1610
    1611
    1612
    1613
    1614
    1615
    1616
    1617
    1618
    1619
    1620
    1621
    1622
    1623
    1624
    1625
    1626
    1627
    1628
    1629
    1630
    1631
    1632
    1633
    1634
    1635
    1636
    1637
    1638
    1639
    1640
    1641
    1642
    1643
    1644
    1645
    1646
    1647
    1648
    1649
    1650
    1651
    1652
    1653
    1654
    1655
    1656
    1657
    1658
    1659
    1660
    1661
    1662
    1663
    1664
    1665
    1666
    1667
    1668
    1669
    1670
    1671
    1672
    1673
    1674
    1675
    1676
    1677
    1678
    1679
    1680
    1681
    1682
    1683
    1684
    1685
    1686
    1687
    1688
    1689
    1690
    1691
    1692
    1693
    1694
    1695
    1696
    1697
    1698
    1699
    1700
    1701
    1702
    1703
    1704
    1705
    1706
    1707
    1708
    1709
    1710
    1711
    1712
    1713
    1714
    1715
    1716
    1717
    1718
    1719
    1720
    1721
    1722
    1723
    1724
    1725
    1726
    1727
    1728
    1729
    1730
    1731
    1732
    1733
    1734
    1735
    1736
    1737
    1738
    1739
    1740
    1741
    1742
    1743
    1744
    1745
    1746
    1747
    1748
    1749
    1750
    1751
    1752
    1753
    1754
    1755
    1756
    1757
    1758
    1759
    1760
    1761
    1762
    1763
    1764
    1765
    1766
    1767
    1768
    1769
    1770
    1771
    1772
    1773
    1774
    1775
    1776
    1777
    1778
    1779
    1780
    1781
    1782
    1783
    1784
    1785
    1786
    1787
    1788
    1789
    1790
    1791
    1792
    1793
    1794
    1795
    1796
    1797
    1798
    1799
    1800
    1801
    1802
    1803
    1804
    1805
    1806
    1807
    1808
    1809
    1810
    1811
    1812
    1813
    1814
    1815
    1816
    1817
    1818
    1819
    1820
    1821
    1822
    1823
    1824
    1825
    1826
    1827
    1828
    1829
    1830
    1831
    1832
    1833
    1834
    1835
    1836
    1837
    1838
    1839
    1840
    1841
    1842
    1843
    1844
    1845
    1846
    1847
    1848
    1849
    1850
    1851
    1852
    1853
    1854
    1855
    1856
    1857
    1858
    1859
    1860
    1861
    1862
    1863
    1864
    1865
    1866
    1867
    1868
    1869
    1870
    1871
    1872
    1873
    1874
    1875
    1876
    1877
    1878
    1879
    1880
    1881
    1882
    1883
    1884
    1885
    1886
    1887
    1888
    1889
    1890
    1891
    1892
    1893
    1894
    1895
    1896
    1897
    1898
    1899
    
```



```
80      2      END;

      1      /* CRLF: */
      CRLF: PROCEDURE ;

      /* SEND CARRIAGE RETURN AND LINE FEED TO THE CRT */

      2      CALL SEND$CHAR$CRT(CR);
      2      CALL SEND$CHAR$CRT(LF);
      2      END;

85      1      SEND$STRING$CRT: PROCEDURE (STRING$ADDRESS);

      2      /* SEND MESSAGE AT STRING$ADDRESS UNTIL '$' IS
      2      DETECTED */
      2      DCL STRING$ADDRESS ADDRESS;
      2      DCL TEMPCHAR BASED STRING$ADDRESS BYTE;
      2      DO WHILE TEMPCHAR <> '$';
```



```
89      3      CALL SEND$CHAR$CRT(TEMPCHAR);
90      3      STRING$ADDRESS=STRING$ADDRESS + 1;
91      3      END;
92      2      END;

93      1      PRINT$TO$CRT: PROCEDURE (STRING$ADDRESS) ;

          2      /* PRINT A STRING TO THE CRT */
          2      DCL STRING$ADDRESS ADDRESS;
          2      CALL CRLF;
          2      CALL SEND$STRING$CRT(STRING$ADDRESS);
          2      CALL CRLF;
          2      END;

94      2
95      2
96      2
97      2
98      2

99      1      READ$CRT$CHAR: PROCEDURE BYTE;
100     2      /* READ A CHARACTER FROM THE CRT */
          2      DCL CHAR BYTE;
```



```
101 2 CHAR= INPUT(246) AND 07FH;  
102 2 IF CRTECHO OR (STATE=RECEIVE) THEN  
103 2 CALL SEND$CHAR$CRT(CHAR);  
104 2 RETURN CHAR;  
105 2 END;  
  
106 1 READ$CHAR$PDP:PROCEDURE BYTE;  
  
107 2 /* READ A CHARACTER FROM THE PDP */  
108 2 DCL CHAR BYTE;  
109 2 CHAR= INPUT(244) AND 07FH;  
111 2 IF PDPECHO THEN CALL SEND$CHAR$PDP(CHAR);  
112 2 /* CONVERT LOWER TO UPPER CASE */  
113 2 IF (CHAR >=61H) AND (CHAR<=7AH) THEN  
114 2 RETURN CHAR AND 0DFH;  
115 2 RETURN CHAR;  
116 2 END;
```



```
115 1      GET$CHAR$CRT$BUF: PROCEDURE BYTE;
      /* GET A CHARACTER FROM THE CRT BUFFER */
116 2      DCL CHAR BYTE;
      /* GET FIRST CHARACTER */
117 2      CHAR =CRT$BUFFER(CRT$BUF$FIRST);
118 2      IF (CRT$BUF$FIRST=CRT$BUF$FIRST+1)>LAST(CRT$BU
      FFER)
120 2      THEN CRT$BUF$FIRST=0; /* WRAP AROUND */
121 2      CRT$BUF$NUMBER=CRT$BUF$NUMBER - 1;
122 2      RETURN CHAR;
      END;

123 1      GET$CHAR$PDP$BUF: PROCEDURE BYTE;
      /* GET A CHAR FROM THE PDP BUFFER */
124 2      DCL CHAR BYTE;
      /* GET FIRST CHARACTER */
125 2      CHAR=PDP$BUFFER(PDP$BUF$FIRST);
```



```

126      2      IF (PDP$BUF$FIRST:=PDP$BUF$FIRST + 1)>LAST$LOC$
PDP$BUF
128      2      THEN PDP$BUF$FIRST=0;
129      2      PDP$BUF$NUMBER=PDP$BUF$NUMBER - 1;
130      2      RETURN CHAR;
      END;

131      1      PUT$CHAR$CRT$BUF: PROCEDURE (CHAR);

      /* A CHAR IS PUT IN THE CRT BUFFER */
132      2      DCL CHAR BYTE;
133      2      IF (CRT$BUF$LAST:=CRT$BUF$LAST + 1)> LAST (CRT$BU
FFER)
135      2      THEN CRT$BUF$LAST=0;
136      2      IF CRT$BUF$NUMBER= 0 THEN
137      2          CRT$BUF$LAST=CRT$BUF$FIRST;
138      2      CRT$BUFFER (CRT$BUF$LAST)=CHAR;
139      2      CRT$BUF$NUMBER=CRT$BUF$NUMBER+1;
140      2      ALARM=0; /* SET FOR NO ALARM FOR NEXT PASS */
      END;

```



```
141 1      PUT$CHAR$PDP$BUF: PROCEDURE(CHAR);
      2
142 2      /* PUT CHARACTER IN PDP BUFFER */
143 2      DCL CHAR BYTE;
      BUF
      IF(PDP$BUF$LAST:=PDP$BUF$LAST + 1)>LAST$LOC$PDP$
      THEN PDP$BUF$LAST=0;
145 2      IF(PDP$BUF$NUMBER=0) THEN
146 2          PDP$BUF$LAST=PDP$BUF$FIRST;
147 2      PDP$BUFFER(PDP$BUF$LAST)=CHAR;
148 2      PDP$BUF$NUMBER=PDP$BUF$NUMBER+1;
149 2      END;

150 1      CRT$BUF$FULL: PROCEDURE BYTE;

151 2      /* CHECK IF CRT BUFFER OVERFLOW */
152 2      RETURN CRT$BUF$NUMBER=LENGTH(CRT$BUFFER);
      END;
```



```
153 1      PDP$BUF$FULL: PROCEDURE BYTE;

154 2      /* CHECK FOR PDP BUFFER OVERFLOW */
155 2      RETURN PDP$BUF$NUMBER=LENGTH$PDP$BUF;
      END;

156 1      PRINT$HEX$NUMBER: PROCEDURE (CHAR);

157 2      /* PRINT HEXADECIMAL COUNTER */
158 2      DCL CHAR BYTE;
159 2      IF CHAR>9 THEN CALL SEND$CHAR$CRT(CHAR-10+'A');
160 2      ELSE CALL SEND$CHAR$CRT(CHAR+'0');
161 2      END;
```



```
162 1      FORMAT$HEX :PROCEDURE (CHAR);
      /* FORMAT DECIMAL NUMBER FOR HEXADECIMAL OUTPUT */
163 2      DCL CHAR BYTE;
164 2      CALL PRINT$HEX$NUMBER(SHR(CHAR,4));
165 2      CALL PRINT$HEX$NUMBER(CHAR AND 0FH);
166 2      END;

167 1      PRINT$CHAR$COUNT: PROCEDURE ;

      /* PRINT DECIMAL COUNTER IN HEXADECIMAL FORMAT */
168 2      CALL CRLF;
169 2      CALL SEND$STRING$CRT(.( 'CHARACTER COUNT:$' ));
170 2      IF D1 <> 0 THEN CALL FORMAT$HEX(D1);
172 2      IF D2 <> 0 THEN CALL FORMAT$HEX(D2);
174 2      CALL FORMAT$HEX(D3);
175 2      END;
```



```

176 1 INIT$CHAR$COUNT: PROCEDURE;
    2
177 2 /* CLEAR DECIMAL COUNTER */
178 2 D3,D2,D1=0;
    2 END;

179 1 COUNT$CHAR: PROCEDURE ;
    2
180 2 /* INCREMENT DECIMAL COUNTER */
181 2 D3=DEC(D3+1);
182 2 D2=DEC(D2 PLUS 0);
183 2 D1=DEC(D1 PLUS 0);
    2 END;

184 1 INIT$NEUTRAL$STATE: PROCEDURE;
    2 /* INITIALIZE NEUTRAL STATE */

```



```
185 2 STATE=NEUTRAL;
186 2 CALL PRINT$TO$CRT(.( 'SYSTEM IN NEUTRAL STATE: $
    ');
187 2 CALL CRLF;
188 2 CALL SEND$CHAR$PDP(CR);
189 2 END;

190 1 INIT$RECEIVE$STATE: PROCEDURE ;

191 2 /* INITIALIZE RECEIVING STATE */
192 2 PDP$BUF$FIRST=0;
193 2 PDP$BUF$LAST=0;
194 2 PDP$BUF$NUMBER=0;
195 2 PREVCHAR='';
196 2 STATE=RECEIVE;
197 2 CALL PRINT$TO$CRT(.( 'SYSTEM IN RECEIVE STATE: $'
    ));
198 2 CALL CRLF;
199 2 CALL INIT$CHAR$COUNT;
200 2 SKIP$FIVE = 0;
    CALL SEND$CHAR$PDP(CR);
```



```
201 2      END;

202 1      INIT$TRANSMIT$STATE: PROCEDURE;

203 2      /* INITIALIZE TRANSMIT STATE */
204 2      EOFILE=0;
205 2      ECHOCHAR=1;
206 2      SENT$ACK = FALSE;
207 2      SAVECHAR='';
208 2      STATE=TRANSMIT;
209 2      CALL PRINT$TO$CRT(.( 'SYSTEM IN TRANSMIT STATE:
    2      $' ));
210 2      CALL CRLF;
211 2      STOP$FLAG=FALSE;
212 2      CALL INIT$CHAR$COUNT;
    2      END;

213 1      BREAK$STATE: PROCEDURE ;
```



```
/* INTERRUPT THE PDP */
DCL I BYTE;
DO I = 1 TO 3;
    CALL SEND$CHAR$PDP(RUBOUT);
END;
END;
```

```
214 2
215 2
216 3
217 3
218 2
```

```
219 1      END$R : PROCEDURE ;

/* TERMINATE RECEIVE STATE ,AND RETURN TO NEUTRAL
CALL PRINT$TO$CRT(.( 'RECEIVE STATE TERMINATED B
CALL PRINT$TO$CRT(.( 'NO FILE CREATED AT FLOPPY
/* DISCARD PDP BUFFER CONTENTS */
PDP$BUF$FIRST=0;
PDP$BUF$LAST=0;
PDP$BUF$NUMBER=0;
```

```
STATE */
220 2
Y OPERATOR.$' )
221 2
DISK.$' ));
222 2
223 2
224 2
```



```

225      /* SEND BREAK SIGNAL TO THE PDP */
226      CALL BREAK$STATE;
227      CALL CLOSE(AFT$IN,.CLOSE$STATUS);
228      CALL DELETE(.FILENAME,DELETE$STATUS);
      END;

```

```

229      1      END$T: PROCEDURE ;

      /* TERMINATE TRANSMIT STATE , RETURN TO NEUTRAL STA
230      2      CALL PRINT$TO$CRT(.( 'TRANSMIT STATE TERMINATED
BY OPERATOR.$'
      -
231      2      CALL PRINT$TO$CRT(.( 'PARTIAL FILE CREATED AT PD
P.$' ));
232      2      /* SEND BREAK SIGNAL TO PDP */
233      2      CALL SEND$CHAR$PDP(ACK);
234      2      SENT$ACK = TRUE;
235      2      CALL CLOSE (AFT$IN,.CLOSE$STATUS);
      STATUS);
236      2      CALL CONSOL (.( ':CI:$' ),.( ':VO:$' ),.CONSOL$
      END;

```



```

237 1      WRITE$RECORD$TO$DISK:PROCEDURE;

238 2      /* WRITE ONE RECORD FROM PDPBUF TO DISK */
      CALL WRITE(AFT$IN,.PDP$BUFFER(PDP$BUF$PTR),128,
.WRITE$STATUS)
      -
239 2      CALL WRITE(0,.PDP$BUFFER(PDP$BUF$PTR),128,.WRIT
E$STATUS);
240 2
241 2      PDP$BUF$PTR = PDP$BUF$PTR + 80H;
242 2      IF PDP$BUF$PTR > LAST$LOC$PDP$BUF THEN
243 3          DO;
244 3              PDP$BUF$NUMBER = 0;
245 2              END;
246 2          ELSE PDP$BUF$NUMBER = PDP$BUF$NUMBER - 128;
      END;

```



```
247 1      WRITE$PDP$BUFFER: PROCEDURE;

248 2      /* WRITE ENTIRE PDP BUFFER TO DISK */
249 2      PDP$BUF$PTR = 0002H;
          PDP$BUF$NUMBER = PDP$BUF$NUMBER -3;
/* PAD PDP BUFFER UNTIL PDP$BUF$NUMBER IS A MUL-
   TIPLE OF 128 */
250 2      DO WHILE ((PDP$BUF$NUMBER - 2) AND 0177Q) <>
0;          CALL PUT$CHAR$PDP$BUF(' ');
251 3      END;
252 3
253 2      CALL OPEN(.AFT$IN,.FILE$NAME,2,0,.OPEN$STAT
US);
254 2      DO WHILE PDP$BUF$NUMBER <> 2;
          /* WRITE NEXT 128 BYTE RECORD TO DISK */
255 3      CALL WRITE$RECORD$TO$DISK;
256 3      END;
257 2      CALL CLOSE (AFT$IN,.CLOSE$STATUS);
258 2      PDP$BUF$PTR = 0002H;
259 2      PDP$BUF$LAST,PDP$BUF$FIRST = 0;
260 2      END;
```



```

261 1      /* REBOOT: */
      REBOOT: PROCEDURE;

262 2      /* GO BACK TO MDS OPERATING SYSTEM : ISIS-11 */
      IF MISSINGDATA THEN CALL PRINT$TO$CRT(
      .('NEUTRAL STATE: SYSTEM MAY NOT HAVE RECEIVED ALL PDP
CHAR TRANSMIT
-   TED $'));
264 2      CALL PRINT$TO$CRT(
      .('NEUTRAL STATE AND REBOOTING TO ISIS-11 $'))
; 265 2      CALL CONSOL((':CI:$'),.(':VO:$'),.CONSOL$STATU
S);
266 2      CALL EXIT;
267 2      END;

/* MAIN ROUTINE: *****/

```



```

268 1      MAIN: DO;
269 2      CALL SET$TTY$2400;
270 2      CALL READ (1,.$FILENAME,128,.$BYTE$COUNT,.$READ$S
TATUS);
271 2      CALL WRITE(0,.$FILE$NAME,14,.$WRITE$STATUS);

/* BEGIN TESTING PORTS , AND SELECT THE APPROPRIATE
ACTION */
272 2      /* SET THE PDP BUFFER PARAMETERS */
273 2      LENGTH$PDP$BUF=TOP-BASE;
          LAST$LOC$PDP$BUF=LENGTH$PDP$BUF-1;
/* NOW, THE POINTERS OF THE BUFFERS OF THE CRT AND
THE
274 2      PDP ARE INITIALIZED */
          PDP$BUF$FIRST,
          PDP$BUF$LAST,
          PDP$BUF$NUMBER,
          CRT$BUF$FIRST,
          CRT$BUF$LAST,
          CRT$BUF$NUMBER=0;

```


/* INITIALIZE SYSTEM BY ENTERING THE NEUTRAL STATE

PDP TO CRT AND CRT TO PDP; UNIX(O/S) TYPE COMMU

NICATION. */

MISSINGDATA=FALSE;

ALARM=FALSE;

CALL INIT\$NEUTRAL\$STATE;

/* BEGIN INFINITE LOOP: *****/

DO FOREVER;

/* FIRST CASE INPUT FROM CRT */

IF INPUT\$STATUS\$CRT THEN

DO; /* CRT INPUT */


```

281 4      /* CRT BUFFER FULL */
282 4      IF CRT$BUF$FULL THEN
283 5          DO; /* SOUND ALARM ; 5 BEEPS */
284 5          IF NOT ALARM THEN
285 5              IF (ALARM:=OUTPUT$STATUS$CRT) THEN
286 6              DO;
287 6                  CALL SEND$CHAR$CRT(BELL);
288 6                  CALL SEND$CHAR$CRT(BELL);
289 6                  CALL SEND$CHAR$CRT(BELL);
290 6                  CALL SEND$CHAR$CRT(BELL);
291 6                  CALL SEND$CHAR$CRT(BELL);
292 5              END;
293 4          END; /* SOUND ALARM */
294 5          ELSE /* CRT BUFFER NOT FULL */
295 5              DO;
296 6                  CHAR=READ$CRT$CHAR;
297 7                  DO CASE STATE;
298 7                      /* NEUTRAL STATE */
299 7                      DO;
300 7                          /* CHECK FOR A COMMAND */
301 7                          IF CHAR =CONTROL$T THEN

```



```

298       CALL INIT$TRANSMIT$STATE;ELSE
299   IF CHAR=CONTROL$R THEN
300       CALL INIT$RECEIVE$STATE; ELSE
301   IF CHAR=CONTROL$C THEN
302       CALL REBOOT; ELSE
303   /* NOT A COMMAND */
304   CALL PUT$CHAR$CRT$BUF(CHAR);
END; /* END NEUTRAL */

```

```

305   /* RECEIVING STATE */
306   DO;
307   IF CHAR=CONTROL$C THEN
308       DO;
309       CALL END$R;
310       CALL REBOOT;
311       END;
312   ELSE /* NOT A COMMAND */
313       CALL PUT$CHAR$CRT$BUF(CHAR);
END; /* END RECEIVE */

```



```

313          /* TRANSMIT STATE */
          DO;
          /* NOT A COMMAND */
          CALL PUT$CHAR$CRT$BUF(CHAR);
          END; /* END TRANSMIT */
          END; /* END CASE STATEMENT */
          END; /* END ELSE CRT BUFFER NOT FULL */
          END; /* INPUT FROM CRT */

```

```

319          /* SECOND CASE: INPUT FROM PDP */
          IF INPUT$STATUS$PDP THEN
          DO; /* PUT DATA IN PDP BUFFER UNLESS FULL */
          IF STATE=RECEIVE THEN
          DO; /* RECEIVING DATA */
          /* PDP BUFFER FULL: GENERATE ERROR MESSAGE ,

```



```

323      5      RETURN TO NEUTRAL STATE */
324      5      IF PDP$BUF$FULL THEN
325      6      DO;
326      6      CALL BREAK$STATE;
          CALL PRINT$TO$CRT(
          .('RECEIVING STATE : PDP BUFFER OVERFL
          CALL PRINT$TO$CRT(.('NO FILE CREATED A
          CALL REBOOT;
          END; /* PDP BUFFER FULL */
          /* BUFFER NOT FULL */
          /* GET CHARACTER FROM PDP */
          CHAR=READ$CHAR$PDP;
          /* PDP PROMPTING : TERMINATE RECEPTION */
          IF ((PREVCHAR=LF) OR (PREVCHAR=CR)) AND CHAR=
          THEN
          DO; /* PROMPTING */
          /* WRITE PDP BUFFER TO DISK */
          CALL WRITE$PDP$BUFFER;
          CALL PRINT$TO$CRT(.('PDP PROMPTING:END OF
          RECEPTION.$' )
          );

```



```
335 6      CALL PRINT$TO$CRT(  
      .('PDP BUFFER/FILE WRITTEN TO DISK.$')  
      );  
  
      /* PRINT NUMBER OF CHARACTERS TRANSMITTED  
      FROM PDP BUFFER TO DISK */  
336 6      CALL PRINT$CHAR$COUNT;  
337 6      CALL SEND$STRING$CRT(  
      .('BYTES TRANSMITTED FROM PDP TO FLOPPY  
DISK.$'));  
338 6  
      /*  
      CALL CRLF;  
      /* RECEPTION COMPLETE: RETURN TO NEUTRAL STATE  
      /*  
339 6      CALL INIT$NEUTRAL$STATE;  
340 6      END; /* PROMPTING */  
      ELSE  
341 5      IF CHAR=END$OF$BLOCK THEN  
342 5      DO;  
343 6      CALL WRITE$PDP$BUFFER;  
344 6      CALL PRINT$TO$CRT(  
      .('RECEIVE STATE: RECEIVED BLOCK.$'));  
      /* CRT ACKNOWLEDGE RECEPTION FROM PDP */  
345 6      CALL PUT$CHAR$CRT$BUF(RCVD);  
346 6      CALL PUT$CHAR$CRT$BUF(CR);  
347 6      END;
```



```
348           ELSE /* NOT END OF RECEPTION-DATA */
349           DO;
350             PREVCHAR=CHAR;
351             CALL PUT$CHAR$PDP$BUF(CHAR);
352             /* INCREMENT NUMBER OF CHARS RECEIVED */
353             IF (SKIP$FIVE < 4) THEN
354             DO;
355               SKIP$FIVE = SKIP$FIVE + 1;
356             END;
357             ELSE
358             DO;
359               CALL COUNT$CHAR;
360             END;
361             END;
362           END; /* END RECEIVING */
363           ELSE /* NEUTRAL OR TRANSMIT STATE */
364           DO;
365             /* PDP BUFFER FULL */
366             IF PDP$BUF$FULL THEN MISSINGDATA=1;
367             /* BUFFER NOT FULL */
368             ELSE
```



```
363 DO;
364 CHAR=READ$CHAR$PDP;
365 IF SAVECHAR=CHAR THEN ECHOCHAR=1;
366 CALL PUT$CHAR$PDP$BUF(CHAR);
367
368 END;
369 END; /* NEUTRAL OR TRANSMIT STATE */
370 END; /*PUT DATA IN PDP BUFFER UNLESS FULL */
```

```
552 */
371 IF OUTPUT$STATUS$CRT THEN
372 /* CAN SEND DATA TO CRT */
373 DO;
374 IF STATE<> RECEIVE THEN
375 DO; /* CHECK IF SOMETHING FOR THE CRT */
376 IF PDP$BUF$NUMBER > 0 THEN
377 CALL SEND$CHAR$CRT(GET$CHAR$PDP$BUF);
378 END;
379 /* ELSE (RECEIVING) DATA FROM PDP TO DISK NOT TO
380 CRT */
```



```

378 4      END; /* CAN SEND DATA TO THE CRT */

      /*FOURTH CASE: SEND DATA TO PDP */
      IF OUTPUT$STATUS$PDP THEN
379 3      DO; /* SEND DATA TO PDP */
380 3      IF STATE <> TRANSMIT THEN
381 4      DO; /* CHECK IF CRT HAS SOMETHING FOR THE PDP */
382 4      IF CRT$BUF$NUMBER > 0 THEN
383 5      CALL SEND$CHAR$PDP(GET$CHAR$CRT$BUF);
384 5      END; /* SEND DATA TO PDP */
385 5      ELSE /* TRANSMITTING DATA */
386 4      DO;

387 5      CALL OPEN (.AFT$IN,.FILE$NAME,1,0,.OPE
N$STATUS);
388 5      BYTE$COUNT=1;
389 5      CALL SEND$CHAR$PDP(CR);
390 5      DO WHILE (BYTE$COUNT<>0 AND STOP$FLAG = FALSE)
;
```



```

391 6
392 6
393 7
394 7
395 7
396 8
397 8
398 8
399 7
400 6
401 6
402 7
403 7
404 8
.BYTE$COUNT,
405 8
,.CONSOL$STATU
- S);
406 8
T,.WRITE$STATU
- S);
407 8
.CONSOL$STATUS
- );
IF INPUT$STATUS$CRT THEN
DO;
CHAR$TWO=READ$CRT$CHAR;
IF CHAR$TWO=CONTROL$C THEN
DO;
CALL END$T;
STOP$FLAG = TRUE;
END;
END;
IF INPUT$STATUS$PDP THEN
DO;
IF (CHAR:=READ$CHAR$PDP) = '+' THEN
DO;
CALL READ(AFT$IN,.PDP$BUFFER,128,
.READ$STATUS);
CALL CONSOL (.(':CI:$'),.(':TO:$')
CALL WRITE(Ø,.PDP$BUFFER,BYTE$COUN
CALL CONSOL(.(':CI:$'),.(':VO:$'),

```



```

408      8      T,..WRITE$STATU
      -      S);
409      8      CHAR='';
410      8      IF ((BYTE$COUNT>0) AND (BYTE$COUNT
<128)) THEN
411      8
412      9
413      9      DO;
414      9      CALL SEND$CHAR$PDP(ACK);
415      8      SENT$ACK=TRUE;
416      8      COUNT$TEMP=0;
      0;
417      9      DO WHILE (BYTE$COUNT-COUNT$TEMP)<>
      0;
418      9      CALL COUNT$CHAR;
419      9      COUNT$TEMP=COUNT$TEMP+1;
420      8      END;
421      7      END;
422      6      END;
423      5      END;
424      5      IF(SENT$ACK = FALSE) THEN
425      6      DO;
426      6      CALL SEND$CHAR$PDP(ACK);
427      5      END;
      CALL CLOSE(AFT$IN,.CLOSE$STATUS);

```



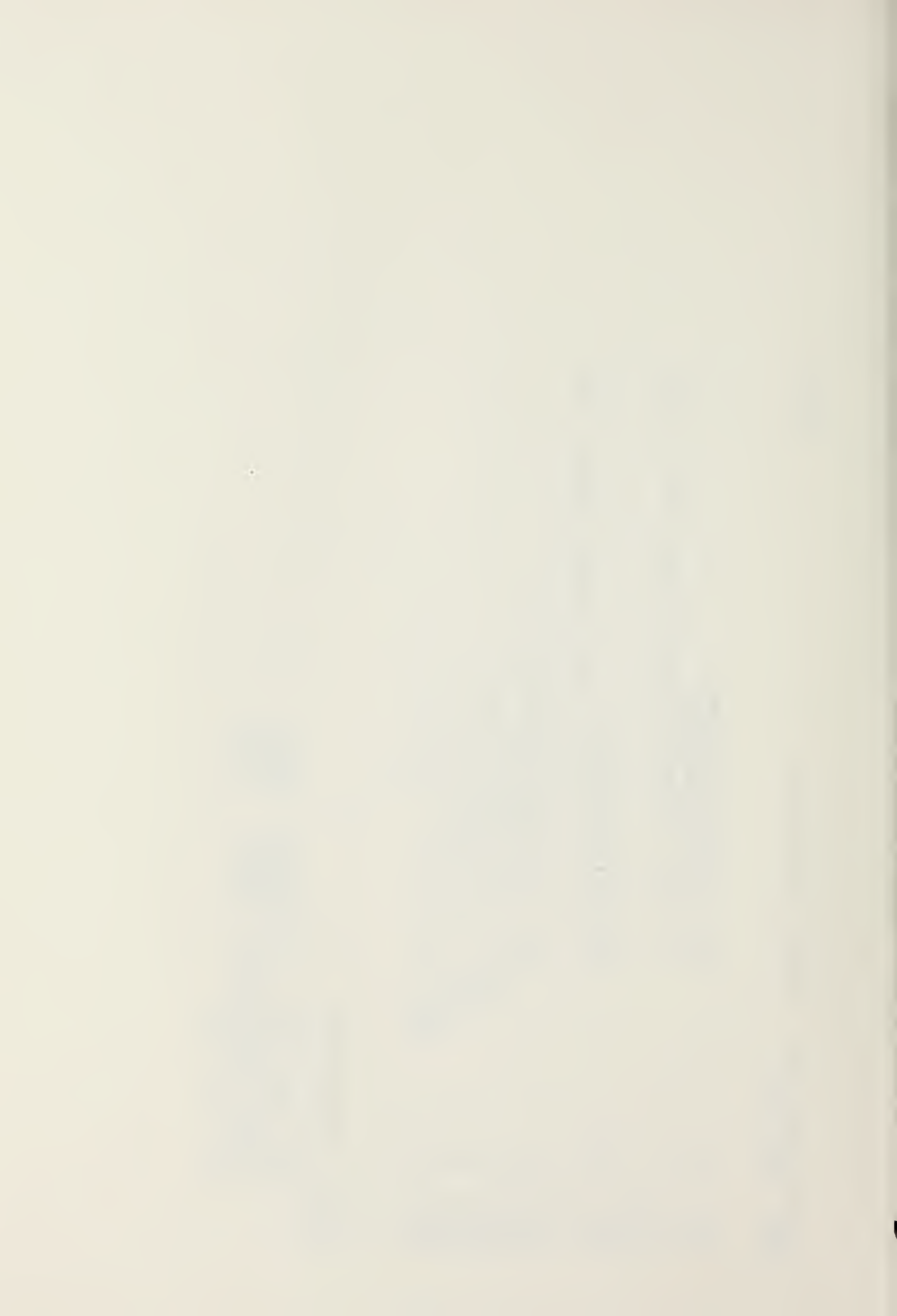
```

428 5      CALL PRINT$CHAR$COUNT;
429 5      CALL SEND$STRING$CRT(
    $' ));
430 5      CALL CRLF;
431 5      CALL PRINT$TO$CRT(
    $' ));
    . $' ));
432 5      CALL INIT$NEUTRAL$STATE;
433 5      END; /*TRANSMIT */
434 4      END; /* END FOURTH CASE */
435 3      END; /* DO FOREVER */
436 2      END; /* END MAIN ROUTINE */
437 1      END; /* H A N D L E R */

```

MODULE INFORMATION:

CODE AREA SIZE	= 0A4CH	2636D
VARIABLE AREA SIZE	= 0190H	400D
MAXIMUM STACK SIZE	= 000AH	10D
835 LINES READ		



MAY 1

PL/M-80 COMPILER MODULE MDSPDP.SRC
979 PAGE 42

Ø PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION

pdpreceive.c

```
int count, fout;
int buf[259];
char c;
int i;
int vl3;
main(argc,argv)
char **argv;
{
    int out();
    signal(2,out);
    if (tcreat(argv[1],buf)==-1)
        { printf("cannot open file");
          exit();
        }
    atty(0,v);
    vl1] =x 0177776;
    vl2] =+ 020;
    stty(0,v);
    printf("+");
    while (1)
    {
        for (i =0;i<128;i++)
        {
            if (( read(0,&c,1))<= 0 ) {printf("+");}
            if(c==0o)
            { fflush(buf);
              printf("+");
              out();
              exit();
            }
            if (c != 015)
```



pdreceive.c

```
    {
       putc(c,buf);
    }
    }
    fflush(buf);
    printf("\n");
}

out()
{
    vl1l = ! 1;
    vl2l = - 020;
    stty(0,v);
    close(buf[0]);
    atty (0,v);
}
```



```

char cr 015;
char lf 012;
char LNB 1;
char RCVN ' ';
char c;
int total 0;
int ibuf [256];
int nr 0;
int nmax 80;
int fd;

main(argc,argv) int argc; char **argv;
{
    argv++;
    if((fd=open(*argv,0))<0)
        {eot();
          printf("cannot open %s0, *argv);
          exit();
        }

    ibuf[0]=fd;
    ibuf[1]=0;
    ibuf[2]=0;

    while(1)
        {if (c== -1)      {eot(); c = 0; argv--; break;}
          if(c== ' ');

```



putsend.c

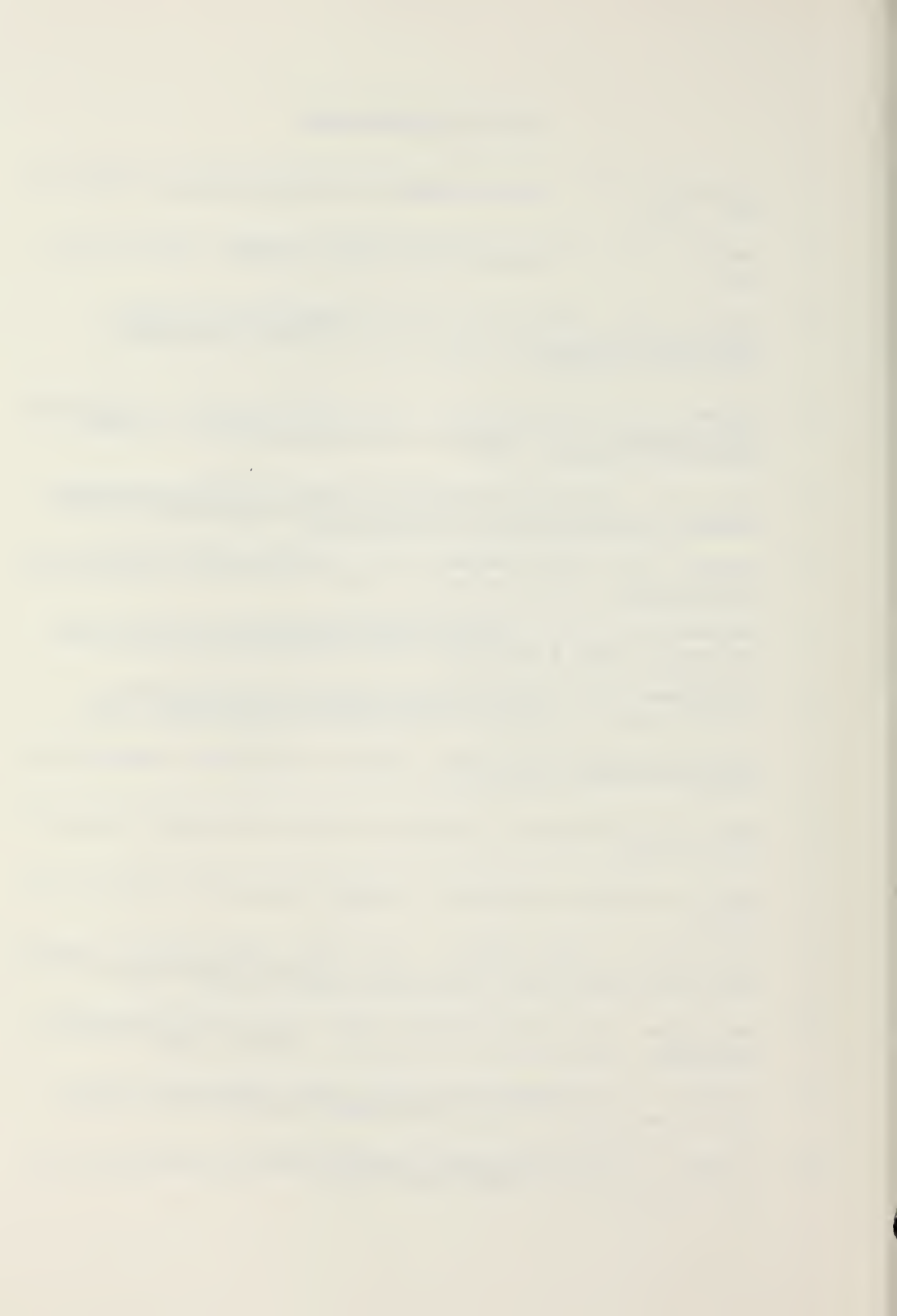
```
{if(nr>=nrmax)
  {putchar(FUR);
   while(getchar() != RCVD);
   nr=0;
  }
  else nr++;
  putchar(c);
  total++;
  c =getc(ibuf);
}
close(fd);
}

eor()
{putchar(cr);
 putchar(lf);
 putchar('%');
}
```

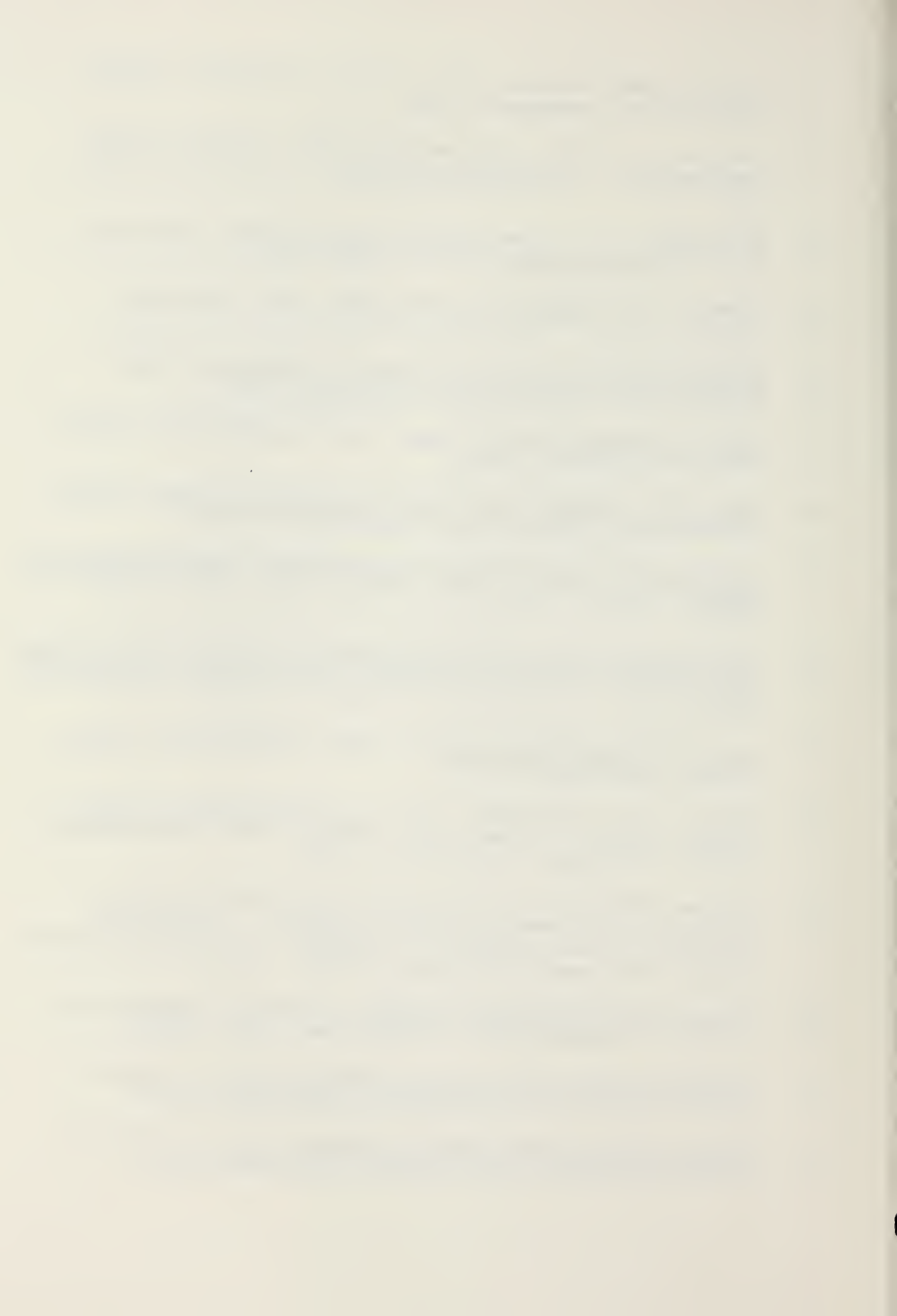


LIST OF REFERENCES

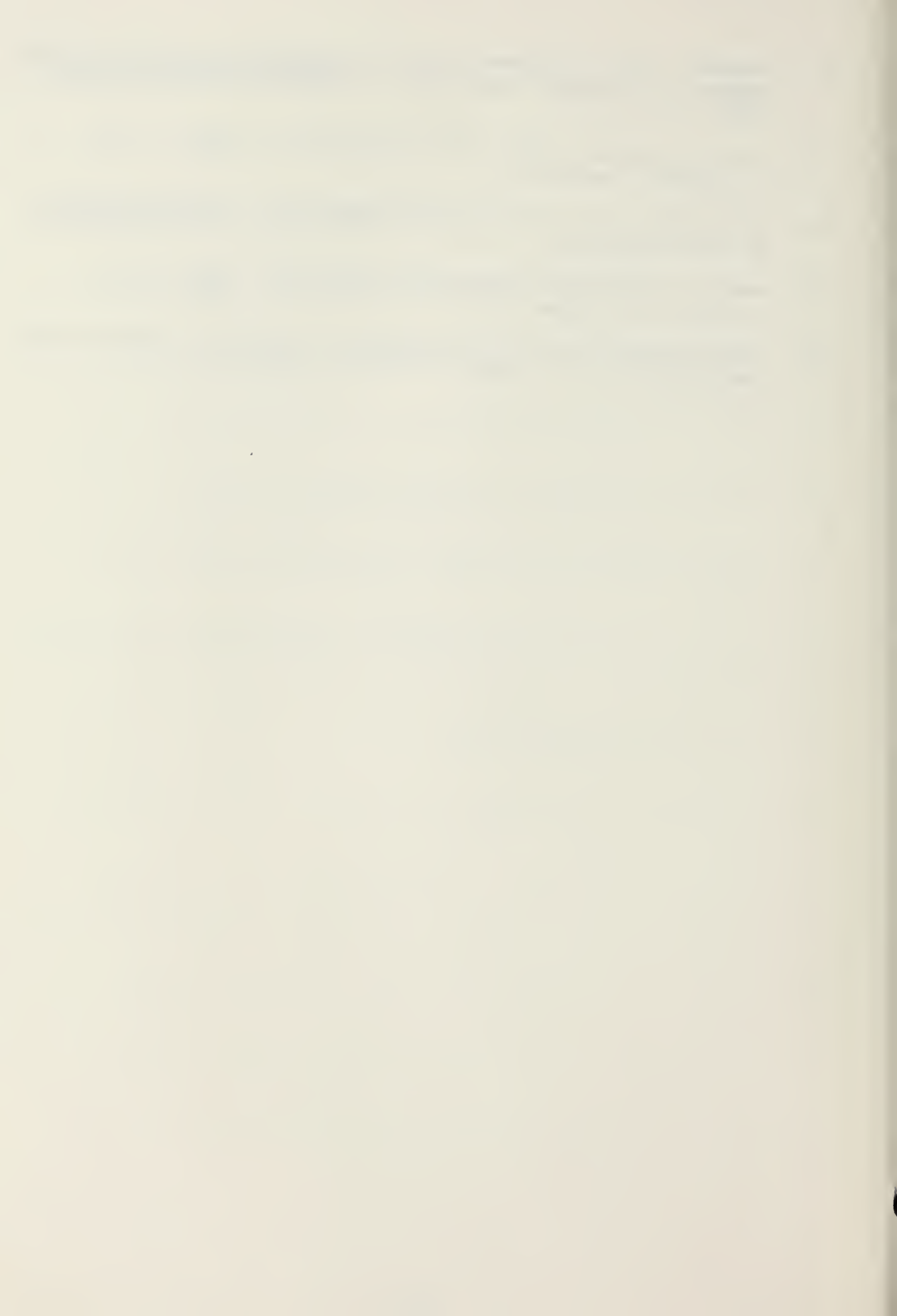
1. Klingman, E.E., Microprocessor Systems Design, Prentice-Hall, 1977.
2. Peatman, J.B., Microcomputer-Based Design, McGraw-Hill, 1977.
3. Morris, R.L. and Miller, J.R., Designing with TTL Integrated Circuits, McGraw-Hill Texas Instruments Electronic Series, '76.
4. Weller, W.J. and Shatzel, A.V. and Nice, H.Y., Practical Microcomputer Programming: The Intel 8080, Northern Technology Books, 1976.
5. Korn, G.A., Microprocessors and Small Digital Computer Systems for Engineers & Scientists, McGraw-Hill, 1977.
6. Hughes, J.K. and Michton, J.I., A Structured Approach to Programming, Prentice-Hall, 1977.
7. Tausworthe, R.C., Standardized Development of Computer Software, Part 1 Methods, Prentice-Hall, 1977.
8. Tanenbaum, A.S., Structured Computer Organization, Prentice-Hall Series in Automatic-Computation, 1976.
9. PDP11 Processor Handbook, Digital Equipment Corporation, 1978.
10. PDP 11/45 Processor Handbook, Digital Equipment Corporation, 1974.
11. PDP 11 Software Handbook, Digital Equipment Corporation, 1978.
12. Thompson, K. and Ritchie, D.M., Unix Programmer's Manual Sixth Edition, Bell Telephone Laboratories, 1975.
13. Kernighan, B.W. and Ritchie, D.M., The "C" Programming Language, Prentice-Hall Software Series, 1978.
14. Lions, J., A Commentary on the Unix Operating System, the University of New South Wales, 1977.
15. Lions, J., Unix Operating System Source Code Level Six, the University of New South Wales.



16. Digital Specification SPO 14.35.10, Software Product Description, September 1978.
17. Powell, D., "PDP-11 Upgrade Path Now Includes 32-bit Minicomputer," Minicomputer News, v. 3, p. 1, 24 Nov 1977.
18. Whitmasch, J., "Intel Bubble Memory Packs 1M Bit on Chip," Computerworld, p. 1, 7 May 1979.
19. Prokop, J., Computers in the Navy, Navy Institute Press, 1976.
20. ISIS-11 PL/M-80 Compiler Operator's Manual, Intel Corporation, Santa Clara, California 1977.
21. ISIS-11 System User's Guide, Intel Corporation, Santa Clara, California, 1978.
22. 8080/8085 Assembly Language Programming Manual, Intel Corporation, Santa Clara, California, 1978.
23. Intellec 800 Microcomputer Development System Operator's Manual, Intel Corporation, Santa Clara, California, 1975.
24. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1976.
25. PL/M 80 Programming Manual, Intel Corporation, Santa Clara, California, 1976.
26. Babin, O.P. and Seaman, R.S., A Microcomputer Based Plasma Display System, M.S. Thesis, Naval Postgraduate School, Monterey, California, 1978.
27. Soares, Goncalves, A.P. De La Cuba, Bravo, J.E., A Microcomputer Based Shipboard Surface - Subsurface Contact Plotter System, M.S. Thesis, Naval Postgraduate School, Monterey, California 1978.
28. Elite 2500 Instruction Manual, Datamedia Corporation, 7300 N. Crescent Blvd., Pennsauken, N.J., 08110.
29. Plasma Display Set Technical Manual Vol. I, Sceince Applications Inc., San Diego, California, 1976.
30. Plasma Display Set Technical Manual Vol. II, Sceince Applications Inc., San Diego, California, 1976.

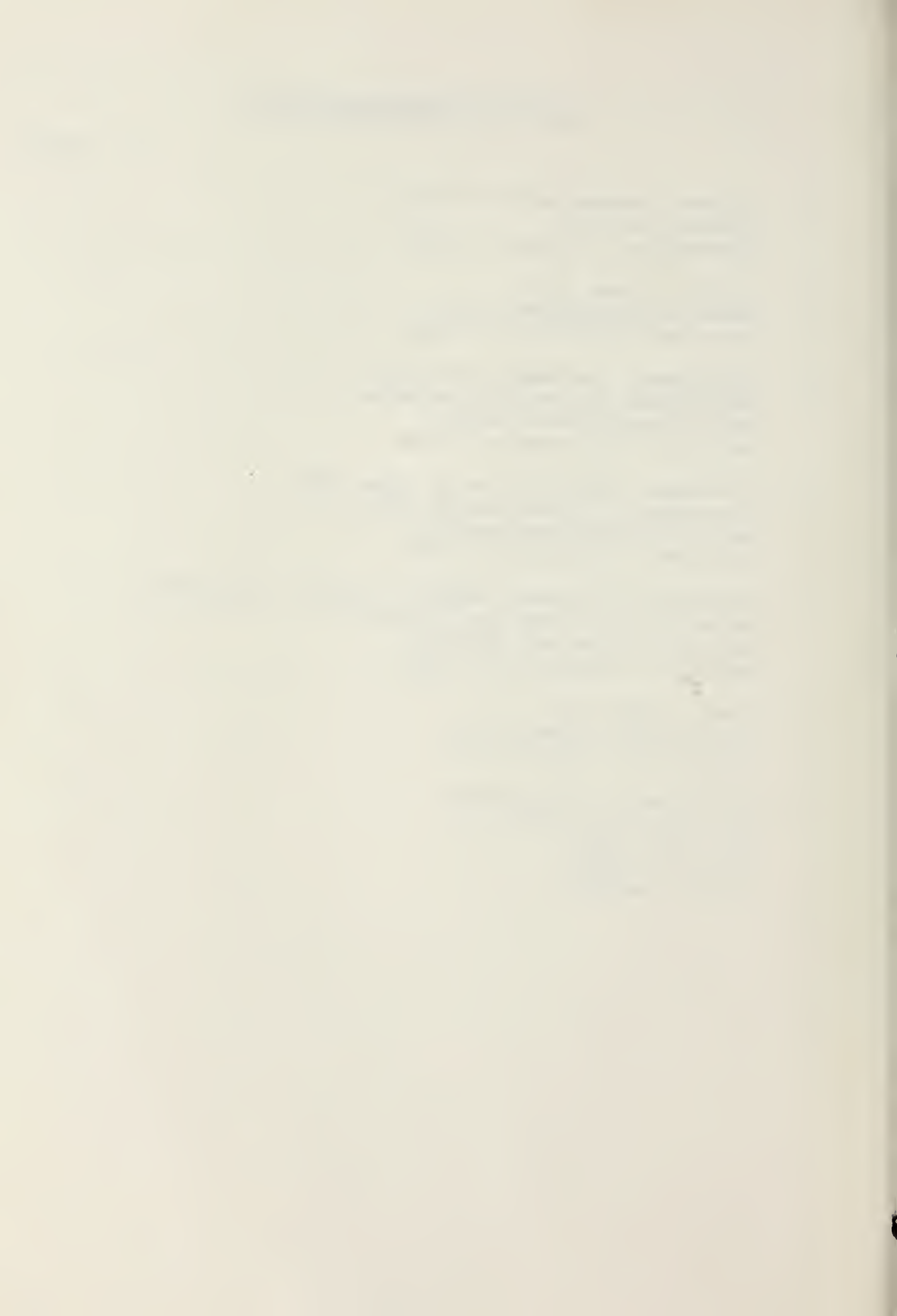


31. Newman, W.M. and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw-Hill Computer Science Series, 1973.
32. Travis, Atkins, "What is an Interrupt," Byte, v. 4, p. 230-236, March 79.
33. Wildelitz, K.S., "Data Base Management," Microcomputing, p. 54-57, May 79.
34. Weems, "Designing Structured Programs," Byte, v. 3, p. 143-156, August 78.
35. EIA Standard RS-232-C, Electronic Industries Association, Washington, D.C., 1969.



INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
4. Professor George A. Rahe, Code 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	2
5. Associate Professor Roger R. Schell, Code 52SJ Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
6. Ivan N. Hall Jr. 3515 Willow Green Court Oakton, Va 22124	6
7. Francisco J. Mariategui Av. Las Artes 1249 URB. San Borja <u>LIMA 34 - PERU</u> South America	6



Thesis
M34265 Mariategui
c.1 Microcomputer based
interactive display
system.

182477

19 AUG 83

20390
279181

Thesis
M34265 Mariategui
c.1 Microcomputer based
interactive display
system.

182477

thesM34265

Microcomputer based interactive display



3 2758 0Q2 12772 2

DUDLEY KNOX LIBRARY